Université de Montréal

Génération de vecteurs de test pour les circuits combinatoires, séquentiels et mixtes basée sur le OBDD

par

Bechir AYARI

Département de Génie Electrique et de Génie Informatique

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSPHIÆ DOCTOR (Ph. D.)
(GENIE ELECTRIQUE)
Mars 1996

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

Génération de vecteurs de test pour les circuits combinatoires, séquentiels et mixtes basée sur le OBDD

présentée par: Bechir AYARI

en vue de l'obtention du diplôme de: Philosophiæ Doctor (Ph.D.)

a été dûment acceptée par le jury d'examen constitué de:

M. SAVARIA Yvon, Ph.D., président

Mme. KAMINSKA Bozena, Ph. D., membre et directeur de recherche

M. ASIM ALKHALILI, Ph. D., membre

M. BOIS GUY, Ph. D., membre

Je dédie cette thèse à ma mère, à mon père, à ma femme et à toute ma famille.

REMERCIEMENTS

Nous tenons à remercier ceux et celles qui, de près ou de loin, ont contribué à ce travail, dont:

Mme Bozena Kaminska, dont la direction a été fondamentale dans l'élaboration de cette thèse, le Département de Génie Electrique et de Genie Informatique de l'Ecole Polytechnique de Montréal; Les membres de jury qui ont accepté d'évaluer le contenu de cette thèse dont Mr. Yvon Savaria, Mr. Asim Alkhalili et Mr. Guy Bois;

Ma femme Basma, ma fille Nihel, ma fille Mariam et l'ensemble de ma famille pour leur soutien, leurs encouragements et leurs prières tout au long de mes études; Naim Ben Hamida qui a largement contribué à nos travaux et avec qui nous avons rédigé plusieurs articles dont deux décrits dans cette thèse; Samir Lejmi et Adel Belhaouane; tous ceux qui ont eu de la gentillesse, aux moments critiques, de céder une station, répondre à une question, etc...

SOMMAIRE

Cette thèse a comme objectif le développement de nouvelles méthodes de test pour les circuits combinatoires, séquentiels et mixtes. Nous montrerons comment, en adoptant des approches algébriques basées sur les diagrammes de décisions binaires (BDD), cet objectif a été atteint.

Dans le Chapitre 2, nous décrirons notre méthode pour manipuler des fonctions logiques quelconques qui est basée sur l'expansion de Shannon et sur les BDD.

Le chapitre 3 décrit deux méthodes algébriques différentes de génération de vecteurs de test pour les circuits combinatoires. Elles sont basées sur la manipulation de fonctions logiques en utilisant les BDD. Elles sont basées aussi sur le modèle de pannes collée-à (stuck-at). La première méthode utilise un ordonnancement unique pour toutes les sorties primaires d'un circuit alors que la deuxième utilise un ordonnancement variable pour chaque sortie.

Dans le quatrième chapitre, nous présenterons une méthode hiérarchique de génération de vecteurs de test pour les circuits combinatoires. Elle est basée sur les BDD. Nous montrerons comment nous avons utilisé la décomposition et la notion de *supergates* pour accélérer la génération de vecteurs de test.

Dans le chapitre 5, nous présenterons notre méthode de génération de vecteurs de test pour les circuits mixtes composés d'un circuit analogique, d'un circuit de conversion analogique-numérique quelconque, et d'un circuit numérique.

Dans le chapitre 6, nous introduirons notre méthode algébrique de génération de vecteurs de test pour les circuits séquentiels. La méthode applique d'abord des vecteurs aléatoires pour réduire le nombre de pannes à considérer dans la génération déterministe.

ABSTRACT

In this thesis, we will present new algebraic approaches to generate test vectors for combinational, sequential, and mixed-signal circuits. These approches are based on function manipulations using reduced binary decision diagram representations (ROBDDs).

In Chapter 2, a new method for boolean function manipulation is presented, which is based on ROBDDs. We present also in this chapter the modifications we have done in order to generate ROBDDs for sequential circuits.

Chapter 3 presents our automatic test generator, called BDD_FTEST, which uses ROBDDs to find a set of test vectors for single stuck lines. By using an ordering strategy based on the network topology for each primary output, we have been able to carry out test vectors generation for a larger set of networks than existing methods based on BDDs.

In Chapter 4, we introduce a hierarchical test generation technique based on OBDD representation and on a new observability concept. Using this concept, we have found that all the undetected faults related to a gate can be studied with almost no more effort than studying only one fault. The concept of dominance is used for accelerating test generation.

In Chapter 5, we propose first a graph modeling for mixed-signal circuits. Second, based on this modeling, we present an automatic test vector generator. In this chapter, only the case of an analog block followed by a digital block is studied. The conditions imposed by each block are taken into account during test vector generation.

We introduce in Chapter 6 a new ATPG (Automatic Test Pattern Generator) for sequential ciruits. To reduce the number of faults to be considered by our algebraic approach, a number of random vectors are applied first. For deterministic test pattern generation, we use algebraic approach based on ROBDD representation.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1

INTRODUCTION

L'utilisation d'outils automatiques pour la conception de circuits intégrés a permis de réduire de façon appréciable le temps requis pour réaliser des systèmes de plus en plus complexes. De plus, les améliorations réalisées dans la technologie de fabrication ont rendu possible la réalisation de circuits intégrés contenant des blocs analogiques et des blocs digitaux dans une même puce. Par ailleurs, l'évolution de la technologie vers des niveaux d'intégration très élevés a rendu le test d'un circuit une tâche de plus en plus complexe.

Le test d'un circuit est une expérience dans laquelle un circuit et la réponse correspondante sont analysés pour vérifier si le circuit se comporte correctement. Aujourd'hui, pour avoir des circuits de bonne qualité, c'est-à-dire sans défectuosités, il faut accorder une très grande importance au test. Le coût du test est une des composantes les plus importantes du coût total d'un nouveau produit.

Plusieurs types de tests peuvent être appliqués à un circuit intégré. Les tests paramétriques sont souvent appliqués pour vérifier les caractéristiques électriques des circuits telles que les tensions de seuils, les courants de fuites, etc... Les tests fonctionnels sont généralement appliqués pour vérifier si un circuit fonctionne correctement (selon les spécifications fonctionnelles). Quant aux tests structurels, ils seront appliqués pour tester la structure interne d'un circuit intégré.

Plusieurs méthodes de génération de vecteurs de test ont été développées pour différents types de circuits intégrés. La génération de vecteurs de test dépend évidemment de la complexité du circuit à tester. Elle dépend aussi de plusieurs autres facteurs comme la technologie et le modèle de pannes utilisé.

Le coût de la génération de vecteurs de test dépend principalement de trois facteurs: les frais de production de vecteurs, la qualité des vecteurs générés (mesurée disons par la couverture de pannes[1]), ainsi que le temps CPU et la mémoire requis par un testeur pour l'application des vecteurs.

Dans le reste de ce chapitre, nous introduirons des notions de base sur la modélisation de défectuosités qui peuvent exister dans un circuit integré. Par la suite, nous parlerons de différentes méthodes de génération de vecteurs de test, proposées dans la litérature, pour les circuits combinatoires, séquentiels et mixtes. Finallement, nous discuterons brièvement les motivations et l'originalité des travaux présentés dans cette thèse.

1.1 Modèle de pannes

Le but du test est de vérifier si un circuit se comporte correctement, c'est-à-dire sans faire des erreurs. Le concept d'erreur a plusieurs significations. Par exemple, une erreur observée lors du diagnostic peut être vue comme un résultat incorrect d'une opération arithmétique; alors que pour un équipement automatique de test, une erreur signifie toujours une valeur binaire incorrecte.

1. Le rapport entre le nombre de pannes que les vecteurs testent et le nombre total de pannes.

Les erreurs observées d'un circuit peuvent avoir plusieurs origines. Elles peuvent être dues à des erreurs de conception comme elles peuvent être dues à des défectuosités de fabrication ou à des défaillances physiques.

Les erreurs de conception peuvent être causées par des spécifications incomplètes ou inconsistantes, ou dues à des violations de règles de conception. Quant aux erreurs apparaissant durant la fabrication, elles peuvent être dues à des connexions incorrectes ou à des courts-circuits. Les défectuosités de fabrication ne sont pas attribuables directement à une erreur humaine, elles résultent plutôt d'une imperfection dans le processus de fabrication. On peut citer l'exemple des courts-circuits et des circuits-ouverts qui sont des défectuosités de fabrication très fréquentes. D'autres défectuosités sont attribuables à des erreurs d'alignement des masques, ou bien à une mauvaise encapsulation.

Quant aux défaillances physiques, elles se produisent durant la durée de vie d'un système. Elles sont généralement dues à l'usure des composants et/ou à l'environnement. En général, les pannes physiques ne permettent pas un traitement mathématique pour générer des vecteurs de test ou pour faire le diagnostic. La solution est de ne considérer que les pannes logiques, qui sont une représentation commode de l'effet des défaillances physiques sur l'opération d'un système. Dans ce cas, une panne physique sera détecté en observant une erreur causée par celui-ci.

Les hypothèses fondamentales concernant les pannes logiques sont référées comme modèle de pannes. Le modèle de pannes le plus utilisé est celui d'une ligne unique "collée" à une valeur logique.

Les pannes logiques représentent l'effet des pannes physiques sur le comportement du circuit modélisé. La question qui se pose est la suivante: quels sont les gains qu'on obtient en modélisant les pannes physiques?

Premièrement, le problème d'analyse de pannes devient un problème logique plutôt que physique. De plus, la complexité de la génération de vecteurs de test sera réduite, puisque plusieurs défectuosités différentes seront modélisées par une même panne. Deuxièmement, il existe des modèles de pannes qui sont indépendants de la technologie. En d'autres termes, le même modèle peut être applicable à plusieurs technologies différentes.

Étant donné une panne et un modèle d'un circuit, on doit être capable, en principe, de déterminer la fonction logique du système en présence d'une panne. Ainsi, la modélisation de pannes est très liée au type de modélisation utilisée pour le circuit. Les pannes définies par un modèle structurel sont appelées des pannes structurelles; leurs effets modifient donc le comportement d'un ciruit en accord avec sa structure et le modèle de panne considéré.

Les pannes fonctionnelles sont définies conjointement avec un modèle de pannes fonctionnelles. Par exemple, l'effet d'une panne fonctionnelle peut être celui de changer la table de vérité d'un composant.

En général, les modèles de pannes logiques supposent que tous les composants sont normaux, c'est à dire, ne contiennent aucune défectuosité, et que seulement leurs interconnexions peuvent être affectées. Les pannes typiques affectant les interconnexions sont des courts-circuits et des circuits ouverts. Un court circuit est formé en connectant des

lignes qui ne sont pas supposées l'être, tandis qu'un circuit ouvert résulte d'une connexion brisée.

Par exemple, dans plusieurs technologies, un court-circuit entre la masse ou l'alimentation et une ligne quelconque du circuit peut amener la ligne à prendre une valeur logique fixe. Autrement dit, la ligne est collée à une valeur logique 0 ou 1. Un court-circuit entre deux lignes quelconques crée toujours une nouvelle fonction logique. Selon le cas, la fonction présentée par le court-circuit peut être un ET câblé (AND bridging fault) ou bien un OU câblé (OR bridging fault).

Dans plusieurs technologies, l'effet d'un circuit-ouvert dans une ligne avec une seule sortance est de supposer que l'entrée, devenue non-connectée, prenne une valeur logique constante, et en conséquence apparait comme une panne du type collée-à.

Notons comment la même panne logique *i collée-à v* ($v \in \{0,1\}$) peut représenter plusieurs défectuosités physiques totalement différentes, *i* circuit-ouvert, *i* court-circuité à la masse ou à l'alimentation, et à n'importe quel composant du circuit qui laisse *i* à la valeur logique *v*.

1.2 Test des circuits combinatoires

Différentes méthodes basées sur le modèle de pannes collée-à ont été proposées pour tester les circuits combinatoires. Pour chacune de ces méthodes on peut associer un coût. Parmi ces méthodes, on trouve la génération aléatoire de vecteurs de test. Cette méthode ne prend pas en considération la fonction ou la structure du circuit à tester.

Pour avoir un test de bonne qualité, on a besoin de simuler un très grand nombre de vecteurs aléatoires. En conséquence, malgré la grande simplicité de la méthode de généra-

tion de vecteurs de test, la détermination de la couverture de pannes par simulation peut être un processus très coûteux. Notons aussi que plus le test est long, plus son application est coûteuse, vu que, d'une part, la taille de la mémoire requise par un testeur sera très grande et que, d'autre part, le temps requis pour faire le test sera, lui aussi, très grand.

En utilisant des méthodes de conception pour la testabilité, les vecteurs aléatoires sont souvent générés en ligne à l'intérieur du circuit lui même. Ceci a comme effet d'augmenter la surface du circuit et d'affecter les performances.

Quant aux générateurs déterministes de vecteurs de test pour les circuits combinatoires, ils ont besoin du modèle du circuit à tester et d'un modèle de pannes. Les tests générés incluent les vecteurs à appliquer et la réponse attendue d'un circuit normal (un circuit non défectueux). Les générateurs peuvent produire aussi d'autres données requises pour localiser les pannes, c'est-à-dire faire le diagnostic. Comparée à la génération aléatoire, la génération déterministe est plus coûteuse mais produit des tests plus courts et de bonne qualité.

La génération de vecteurs de test pour les circuits combinatoires peut être vue comme un problème de "branch-and-bound"[16] [17] [18] [19]. La complexité d'une telle approche est généralement fonction du nombre de retour arrières impliqués.

Le but de n'importe quel générateur basé sur un modèle de pannes est de trouver un vecteur, ou encore une assignation aux entrées primaires, pour tester une panne bien spécifique. Le problème à résoudre lors de la génération d'un vecteur de test pour une panne l *collée-à* v est d'activer la panne, c'est-à-dire de justifier une valeur $\overline{v}$ à la ligne l, et de propager l'erreur résultante de l vers une des sorties primaires.

L'idée principale du *branch-and-bound* est que si un problème ne peut pas être résolu directement, il sera décomposé en sous-problèmes à résoudre en premier. En d'autres termes, on essaye de diviser pour régner. Une propriété de ces algorithmes est qu'ils sont exhaustifs; c'est-à-dire si on leur alloue des resources de calcul illimitées, ils sont garanti de trouver une solution (un test) s'il en existe une. Ce qui fait que, lorsque l'algorithme échoue dans la tâche d'obtenir un test pour une panne, la panne ne peut pas être détectée.

La recherche d'un vecteur de test implique souvent des décisions à prendre. Ceci implique que souvent l'algorithme de test doit choisir une alternative parmi plusieurs. Par exemple, pour propager une erreur, l'algorithme commence par choisir une des alternatives. Mais, en agissant ainsi, il peut prendre une décision qui mènera à un conflit, par exemple il peut assigner à une même ligne deux valeurs logiques différentes. Souvent, une décision implique une assignation à certaines lignes. Pour chaque assignation, il faut en calculer les conséquences et vérifier leurs cohérences avec les valeurs déjà calculées. Cette procédure est référée comme une implication.

Pour permettre une exploration systématique de tout l'espace de recherche, les algorithmes basés sur le *branch-and-bound* utilisent des retour arrières dans la stratégie de recherche. Après chaque décision, ces algorithmes stockent toutes les valeurs assignées pour être capable de les effacer lorsque la décision mènera à une contradiction.

Les algorithmes basés sur le *branch-and-bound* montrent leurs pires performances lorsque le circuit analysé renferme plusieurs pannes redondantes. Dans ce cas, un très grand nombre de backtracking sera nécessaire avant de déclarer une panne redondante. Pour ces méthodes, le meilleur cas aura lieu lorsque le résultat (générer un test ou trouver une redondance) est obtenu sans backtracking. Ceci est équivalent à dire que le résultat est

obtenu seulement par implications ou seules des décisions correctes ont été prises. Ceci est toujours le cas de circuits qui n'ont pas de sortances (fanouts) reconvergeantes. Pour ces types de circuits, aucune décision ne mènera à un conflit et toutes les pannes sont détectables.

Pour baisser le temps total du test à un plafond raisonable, les générateurs de vecteurs de test qu'on trouve en pratique sont limités à se faire qu'une quantité limitée de recherche [16] [17] [18] [19]. Généralement, la recherche sera abandonnée lorsque le nombre de décisions incorrectes (ou le temps d'exécution) atteint une limite, qui est généralement spécifiée par l'utilisateur. Conséquemment, l'algorithme ne générera pas des vecteurs de test pour plusieurs pannes détectables.

PODEM [17], par exemple, est un générateur de vecteurs de test caractérisé par un espace de recherche direct, c'est-à-dire que les décisions ne concernent que l'assignation des entrées primaires. Pour justifier une valeur V_k à une la ligne k, PODEM fixe un objectif (k, V_k) à atteindre, en assignant les entrées primaires du circuit. La procédure utilisé pour le retour arrière transforme un objectif désiré en une assignation qui contribue à arriver à l'objectif.

Plusieurs travaux de recherche ont été faits pour réduire l'espace de solutions et le nombre de retour arrières impliqué [18] [19] [16]. Cependant, dans plusieurs cas, même si l'espace de solution est réduit, le nombre de cas à étudier reste toujours grand.

Les autres techniques de test pour les circuits combinatoires sont basées principalement sur la différence booléenne[1]. Ces techniques permettent de déterminer tout l'ensemble de vecteurs de test pour une panne donnée.

Les méthodes algébriques étaient très lentes par rapport aux méthodes basées sur le branch-and-bound. Récemment, des méthodes algébriques ont été proposées pour la manipulation efficace des fonctions logiques. Elles sont assez rapide pour être utilisées dans la génération de vecteurs de test. Ces méthodes abordent le test des circuits combinatoires différement [20] [21] [22]. Elles sont toutes basées sur la manipulation de fonctions logiques et utilisent, généralement, la différence booléenne pour trouver des vecteurs de test.

La méthode de Larrabee [21] trouve d'abord la différence booléenne en fonction des entrées et des sorties de chaque porte d'un circuit combinatoire. Par la suite, cette méthode essaye de satisfaire la fonction correspondante à la différence booléenne.

Smirat et *al.* [23] ont présenté un algorithme qui détermine un vecteur de test en satisfaisant une équation booléenne dérivée du modèle d'un circuit en ajoutant les conditions nécessaires pour l'activation d'une panne et la sensibilisation d'un chemin.

L'algorithme TSUNAMI [13] est, lui aussi, basé sur une approche algébrique utilisant les diagrammes de décisions binaires (Binary Decision Diagram ou BDD). Il est basé sur la sensibilisation des chemins. Cet algorithme détermine l'ensemble d'assignations permettant de propager une erreur à travers une porte logique vers une des sorties primaires. TSUNAMI arrête le calcul une fois que l'erreur est propagée à une des sorties primaires du circuit ou bien quand elle devient non observable.

---

1. La différence booléenne entre deux fonctions est tout simplement le résultat de l'opération XOR entre ces deux fonctions.

Contrairement aux méthodes basées sur le "branch-and-bound", les méthodes algébriques de génération de vecteurs de test basées sur les diagrammes de décisions binaires, [13] et [25], permettent d'étudier les pannes redondantes aussi facilement que toutes les autres pannes du circuit. Ceci est dû à la manipulation efficace des fonctions booléennes qu'offre le BDD. La limite de ces méthodes réside dans les tailles des BDD à manipuler. Ces dernières sont sensibles à l'ordonnancement donné aux variables de l'entrée du circuit.

Les algorithmes de génération de vecteurs de test basés sur les BDD, décrits dans [13] et [25], utilisent un ordonnancement unique pour toutes les sorties primaires, ce qui est généralement très difficile à obtenir.

Notons que, pour certains circuits combinatoires d'essai [7], on ne connaît pas d'ordonnancement efficace unique pour générer des OBDD à toutes les sorties primaires. Ce qui implique que la génération de vecteurs de test n'est pas possible en utilisant un ordonnancement fixe. Par contre, ceci est possible avec un ordonnancement variable [3].

1.3 Test des circuits séquentiels

La génération de vecteurs de test pour un circuit séquentiel est une tâche beaucoup plus compliquée à réaliser que pour un circuit combinatoire, puisque la réponse d'un circuit séquentiel à une séquence d'entrée dépend de son état initial. Notons aussi que lorsque le circuit est sous alimentation, l'état initial d'un élément de mémoire tel qu'une bascule est toujours inconnu. C'est la raison pour laquelle, avant le début de l'opération normale d'un circuit séquentiel, une séquence d'initialisation sera appliquée pour amener le circuit vers un état.

La plupart des générateurs de vecteurs de test pour les circuits séquentiels utilisent le modèle itératif du circuit séquentiel [50]. Ces méthodes dupliquent le circuit pour prendre en considération plusieurs périodes d'horloge consécutives. De cette manière, les méthodes de génération de vecteurs de test pour les circuits combinatoires peuvent être adaptées aux circuits séquentiels.

Pour traiter l'état initial inconnu, les algorithmes de simulation utilisent une valeur logique distincte, dénotée par X. La valeur logique X est traitée en même temps que les valeurs logiques binaires. L'extension des opérateurs booléens à la logique à 3 valeurs est basée sur le raisonnement suivant: la valeur X représente une valeur dans l'ensemble {0,1}. De même, on peut traiter les valeurs 0 et 1 comme étant les ensembles {0} et {1} respectivement. Une opération logique entre p et q, où $p, q \in \{0,1,X\}$, peut être considérée comme une opération sur les ensembles de valeurs représentant p et q. Elle peut être définie comme l'ensemble union de toutes les opérations binaires entre les composantes des deux ensembles. Par exemple, ET(0,X) = ET{{0}, {0,1}} = {ET(0,0), ET(0,1)}= {0,0} = {0} = 0.

De nombreux chercheurs ont tenté de mettre au point des techniques pratiques de génération de vecteurs de test destinés aux circuits séquentiels [49] [50] [51]. Jusqu'à maintenant, les générateurs automatiques de vecteurs de test ne peuvent être appliqués qu'à des circuits de faible complexité ou à des circuits munis d'un état d'initialisation [47], ou encore à des circuits pour lesquels une séquence de synchronisation peut être trouvée [46].

Récemment Mercer *et al.* [48] ont décrit une autre méthode de génération de vecteurs de test pour les circuits séquentiels qui est basée sur le branch-and-bound. Elle utilise une

technique algébrique basée sur la représentation BDD pour justifier un état d'un circuit séquentiel.

1.4 Test des circuits mixtes

Les améliorations réalisées dans la technologie de fabrication a rendu possible la réalisation de circuits mixtes contenant des blocs analogiques et des blocs digitaux dans une même puce. Cependant, le test de ces circuits sera plus compliqué à réaliser que pour celui d'un circuit purement digital ou un circuit purement analogique.

Différentes techniques ont été proposées pour le test des circuits digitaux et d'autres pour le test des circuits analogiques. Mais, on n'a pas trouvé dans la littérature deux techniques (une pour le test d'un bloc analogique et l'autre pour le test d'un bloc analogique) qui peuvent être jumelées dans une seule permettant le test d'un circuit mixte au complet sans le modifier. Dû à la difficulté de contrôler des signaux digitaux à partir des entrées analogiques et dû à la difficulté d'observer les valeurs prises par les sorties analogiques, le test des circuits mixtes est une tâche très complexe..

Soulignons aussi que des solutions spécifiques ont été développées pour tester des circuits standards, comme, par exemple, les convertisseurs analogiques/numériques, [29] [30].

Une des solutions qui peut être utilisée pour faciliter le test des circuits mixtes consiste à utiliser des techniques de conception pour la testabilité (design for testability techniques). Le circuit mixte sera ou bien dans le mode test ou bien dans le mode normal. En mode test, le circuit est décomposé en différents blocs dont les entrées et les sorties peuvent être accédées facilement.

La décomposition peut être faite en utilisant soit des multiplexeurs, soit le bus standard IEEE P1149.4 [34], ce qui permettra d'avoir une meilleure contrôlabilité et une meilleur observabilité des signaux numériques et analogiques. On peut ajouter aussi que des circuits d'auto-test peuvent être incorporés à l'intérieur de la puce. Ainsi, on n'aura pas besoin d'un testeur pour appliquer des vecteurs au circuit.

Notons que l'isolation des blocs analogiques et numériques durant le mode test a comme effet d'augmenter la surface du circuit mixte, ainsi que le nombre de broches d'entrées/sorties. En utilisant une telle approche, les interconnexions entre les blocs seront difficile à tester.

Alani et *al.* [35] ont développé une méthode qui peut être appliquée pour tester les pannes catastrophiques (hard faults) dans un circuit mixte. Cette méthode apporte d'abord des modifications au circuit original, et avec une analyse de la réponse du circuit au régime permanent on détermine si le circuit est défectueux ou non.

L' autre alternative qui peut être utilisée pour tester les circuits mixtes sans décomposition consiste à appliquer des impulsions pour exciter le circuit et ensuite capturer les réponses transitoires des lignes du circuit à l'aide de convertisseurs analogique/numérique très rapides [36]. Ainsi, en analysant les signaux capturés, la méthode déterminera si le circuit contient ou non des défectuosités.

Toutes les méthodes de génération de vecteurs de test que nous introduirons sont des méthodes algébriques, c'est-à-dire basées sur la manipulation des fonctions booléennes. Donc, nous avons besoin d'une méthode rapide et efficace pour manipuler des fonctions logiques.

En se basant sur les BDD, Karl et *al.* [6] ont décrit un logiciel permettant la manipulation efficace des fonctions booléennes. Celui-ci ne peut pas être appliqué pour générer des BDD pour les circuits séquentiels, à moins qu'on suppose que toutes les bascules sont dans une chaîne de balayage (full scan), en d'autres termes considérées comme étant des entrées primaires.

Puisque l'état initial d'un circuit séquentiel est généralement inconnu. Il faut être capable de manipuler des fonctions partiellement définies pour générer des vecteurs de test pour ces circuits.

Notons aussi qu'il n'est pas possible de faire des modifications sur ce logiciel, puisque tout ce que nous possédons est le code exécutable. Nous avons décidé donc de développer notre propre méthode de manipulation de fonctions logiques.

1.5 Les Objectifs de cette thèse

Les objectifs de cette thèse sont de développer des outils rapides pour tester différents types de circuits. Ce travail vise le développement de nouvelles méthodes efficaces de génération de vecteurs de test pour les circuits combinatoires, séquentiels et mixtes. Comme les méthodes présentées dans [13] et [25], nos méthodes de test pour les circuits combinatoires sont basées sur la représentation BDD. Nos méthodes diffèrent de celles présentées dans [13] et [25] par la stratégie qu'elles utilisent pour générer des vecteurs de test pour des circuits plus complexes, sans avoir recours à des retour arrières.

L'originalité de cette thèse réside dans la nouvelle méthode efficace et rapide que nous avons développé pour manipuler les fonctions logiques (qui peuvent être partiellement définies), dans la stratégie que nous utilisons dans la génération de vecteurs de test

sans retour arrières pour différents types de circuits, ainsi que dans la modélisation que nous utilisons pour générer des tests pour les circuits mixtes.

La thèse est divisée en 7 chapitres. Dans le Chapitre 2, nous présenterons la méthode que nous avons développée pour manipuler efficacement des fonctions logiques. Les chapitres 3 et 4 décrivent deux méthodes de génération de vecteurs de test pour des circuits combinatoires. Le chapitre 5 présente notre méthode de génération de vecteurs de test pour les circuits mixtes. Quant au chapitre 6, il discute la génération de vecteurs de test pour les circuits séquentiels. Les chapitres sont divisés comme suit:

Dans le chapitre 2, nous présenterons les algorithmes que nous avons développé pour manipuler des fonctions logiques. Ces algorithmes acceptent des fonctions partiellement définies, c'est-à-dire dont les valeurs ne sont pas définies pour certaines assignations à leurs variables.

Les algorithmes que nous proposons sont basés sur les diagrammes de décisions binaires (BDD) et sur les algorithmes de Bryant [1], pour lesquels nous avons apporté des modifications majeures pour les accélérer et pour minimiser la taille de la mémoire requise. De plus, Contrairement à l'approche de Bryant, qui nécessite deux étapes pour obtenir un graphe réduit, notre approche génère un BDD réduit en une seule étape.

Le chapitre 3 décrit un générateur de vecteurs de test pour les circuits combinatoires. Ce générateur utilise une méthode algébrique pour trouver un vecteur de test pour chaque panne non-détectée de type collée-à (stuck-at) dans un circuit combinatoire.

Dans ce chapitre, nous présenterons deux approches différentes. La première utilise un ordonnancement unique pour toutes les sorties primaires du circuit considéré, alors que

la deuxième utilise un ordonnancement variable pour chacune des sorties primaires. Nous montrons comment on peut se servir de la différence booléenne et la représentation OBDD [1] pour générer rapidement des vecteurs de test pour les circuits combinatoires sans back-traking. Nous montrerons que la deuxième méthode permet d'étudier plus de circuits que les autres approches basées sur les BDD [13] [25].

Dans le quatrième chapitre, une méthode hiérarchique, basée aussi sur le OBDD, est présentée pour générer aussi des vecteurs de test pour les circuits combinatoires. Nous montrerons comment nous pouvons nous servir de la décomposition et de la notion de *supergates* pour accélérer la génération de vecteurs de test. Nous montrerons aussi comment générer rapidement des vecteurs de test pour toutes les pannes correspondantes à une porte logique en utilisant une nouvelle définition de l'observabilité. L'observabilité est une fonction logique renfermant toutes les assignations possibles des entrées primaires qui permettent d'observer, à la sortie, la valeur prise par une ligne du circuit.

Nous décrirons dans le chapitre 5 l'approche que nous avons développé pour tester les circuits mixtes composés d'un circuit analogique, d'un circuit de conversion analogique-numérique quelconque, et d'un circuit numérique.

Dans ce chapitre, nous présenterons une nouvelle modélisation pour les circuits mixtes. En se basant sur celle-ci, nous démontrerons que l'automatisation de la génération de vecteurs de test est possible pour ces types de circuits, sans avoir besoin d'aucune décomposition.

Notre approche prend en considération les conditions imposées par chacun des blocs d'un circuit mixte durant la génération de vecteurs de test. Pour tester la partie analogique,

nous choisissons les paramètres à mesurer qui permettront une couverture maximale de pannes. Pour le test des circuits numériques, nous utiliserons une méthode algébrique basée sur le modèle de pannes collée-à (stuck-at).

Le travail présenté au chapitre 5 a été fait avec la collaboration de BenHamida Naim qui a été responsable du test des blocs analogiques des circuits mixtes. Chacun d'entre nous a adapté sa méthode de génération de vecteurs de test pour pouvoir tester un circuit mixte sans le décomposer.

Le chapitre 6 démontre comment nous pouvons générer des vecteurs de test pour les circuits séquentiels. Cette méthode apporte quelques modifications sur l'approche que nous avons développé pour générer des vecteurs de test pour les circuits combinatoires. Quant à la génération de vecteurs de test pour un circuit séquentiel, nous utiliserons son modèle itératif.

Nous avons démontré aussi que nous pouvons utiliser la génération aléatoire de vecteurs de test au début. En se basant sur des mesures de testabilité [9] [11], seuls les vecteurs qui détectent des pannes seront retenus. En contre partie, pour les circuits séquentiels, l'ordre des vecteurs de test a une influence sur la couverture de pannes. Donc, lors de la compaction, nous conserverons l'ordre des vecteurs de test pour ne pas affecter la couverture de panne.

En générant aléatoirement des vecteurs de test, le nombre de pannes à considérer dans la génération déterministe de vecteurs de test sera réduit.

Le travail présenté dans le chapitre 6 a été fait aussi avec la collaboration de Naim Ben-Hamida qui a été responsable de la génération et la compaction de vecteurs aléatoires en se basant sur des mesures de testabilité pour les circuits combinatoires.

En conclusion de la thèse, un rappel des points saillants est présenté ainsi qu'une discussion sur la recherche à venir.

CHAPITRE 2

Algorithmes de génération de BDDs réduits pour les circuits combinatoires et séquentiels

Introduction

Le présent chapitre est basé sur un papier de conférence intitulé "*New Efficient Algorithms for Generation of ROBDDs for Combinational and Sequential Circuits*" qui a été soumis à *IEEE Transactions on Computer-Aided-Design*.

Dans plusieurs domaines, tels que la conception et le test, il est très important de représenter et de manipuler efficacement des fonctions booléennes. Les méthodes permettant une manipulation efficace de fonctions booléennes sont celles basées sur les diagrammes de décision binaire, en anglais BDD.

Etant donné que nous ne possédons pas les codes sources de ces méthodes, nous ne pouvons pas faire les modifications qui s'imposent pour pouvoir, par exemple, faire des manipulations de fonctions partiellement définies pour générer des vecteurs de test pour les circuits séquentiels. On entend par fonction partiellement définie une fonction dont la valeur n'est pas définie pour certaines assignations à ses variables d'entrée (les variables dont elle dépend).

Nous avons décidé de développer notre propre méthode pour manipuler des fonctions logiques quelconques qui est basée sur l'expansion de Shannon et sur les diagrammes de décision binaire. Elle est simple, efficace et donne des résultats comparables, et dans plusieurs cas meilleurs, que les autres approches.

Nos algorithmes sont basés sur les algorithmes de Bryant [1], auxquels nous avons apporté des modifications majeures pour les accélérer et leur permettre d'accepter des arcs complémentaires. Notre approche génère un BDD réduit en une seule étape, alors qu'en utilisant l'approche de Bryant, deux étapes seront nécessaires.

Nous montrerons aussi, dans ce chapitre, comment nous avons profité de la programmation en langage C pour représenter efficacement les fonctions booléennes. Dans notre cas, les BDD peuvent être référés par deux types d'arcs: des arcs dits réguliers et des arcs dits complémentaires. De cette façon, deux fonctions complémentaires référeront aux mêmes BDD, mais avec deux arcs différents. Ceci a comme effet d'accélérer nos algorithmes en plus de réduire la taille de la mémoire requise pour stocker les graphes (BDD).

Nous avons trouvé qu'en représentant une fonction logique par un entier, nous pouvons stocker à la fois l'adresse du noeud racine du BDD correspondant et le type d'arc avec lequel ce BDD est référé. Puisqu'un noeud du graphe sera toujours stocké dans une adresse qui est multiple de 4, le bit le moins significatif de l'adresse sera toujours 0. Ce bit peut être utilisé pour contenir l'information sur le type d'arc avec lequel ce noeud est référé. De cette façon, deux fonctions complémentaires seront identifiées facilement.

Nous décrirons aussi, dans ce chapitre, la méthode que nous utiliserons pour générer des BDD pour des circuits séquentiels synchrones. Pour étudier ceux-ci, nous devons

d'abord les modéliser. La modélisation à adopter doit tenir compte du fait que l'état initial d'un circuit séquentiel est toujours inconnu. Aussi, elle doit prendre le temps en considération, puisque l'état présent d'un circuit séquentiel dépend de ses états précédents.

Nous avons trouvé que le modèle itératif répond très bien aux critères mentionnés ci-haut. Ce modèle transforme la dépendance dans le temps en une dépendance dans l'espace. Le problème qui reste à régler est le nombre maximum de périodes d'horloges à considérer. Ce nombre peut être soit spécifié par l'utilisateur ou bien estimé en utilisant des mesures de testabilité pour les circuits séquentiels.

Dans ce chapitre, nous présenterons une méthode qui peut être appliquée pour manipuler des fonctions partiellement définies. Les BDD générés contiendront l'information sur le temps. Pour représenter l'état inconnu, à part les noeuds terminaux 0 et 1, les BDD peuvent contenir un troisième noeud terminal, que nous avons appelé X. De cette manière, nous avons rendu possible la génération de vecteurs de test pour les circuits séquentiels synchrones et les circuits mixtes.

Ce chapitre est organisé comme suit: Dans la section 2.2, les principes fondamentaux des BDD sont révisés. Dans la section 2.3, nous présenterons nos algorithmes de manipulation de fonctions logiques qui sont basés sur les BDD. Pour démontrer l'efficacité de notre approche, nous montrerons à la section 2.4 les résultats expérimentaux obtenus pour des circuits combinatoires et séquentiels d'essai. Nous concluons à la section 2.5.

New Efficient Algorithms for Generation of ROBDDs for Combinational and Sequential Circuits

Ayari Bechir and Bozena Kaminska

Department of Electrical and Computer Engineering

Ecole Polytechnique de Montréal

P.O. Box 6079, Station "Centre-Ville", Montréal, Canada (H3C 3A7)

Abstract:

In this paper, we present a method for efficient boolean function manipulations. It uses a new data structure and an associated set of efficient manipulation algorithms. Our approach is based on the reduced binary decision diagram representation (ROBDD) and on the Shannon expansion. It uses complement edges to reduce graph sizes and computation time. We present also the modifications that are introduced to our ROBDD generation procedures to make it applicable to sequential circuits. The experimental results show that our method gives better performances than the other approaches.

2.1 Introduction

It is important to represent and manipulate boolean functions efficiently in various fields such as logic design, verification and testing. Bryant [1] proposed a unified data structure by ordering the decision variables in the well-known binary decision diagram to

represent Boolean functions. The resultant representation, which will be called ordered binary decision diagram (OBDD), has found many useful applications. Bryant has also presented algorithms to produce reduced OBDDs (ROBDDs) for boolean function manipulations.

Based on the *If_then_else* operator, a BDD software package has been developed [6] to efficiently manipulate boolean functions. The approach uses complement edges to accelerate BDD generation and to reduce BDD sizes.

It is known that the size of an ROBDD is sensitive to the ordering of decision variables. In [2] and [3], the authors have shown that the ROBDD can be optimized by an appropriate choice of variable ordering. Based on the topology of a circuit, the heuristics presented in [2] and [3] lead to a good ordering for the inputs of a circuit.

This paper introduces simple and efficient algorithms for function manipulations. The algorithms are based on the Shannon expansion and ROBDD representation, similarly in [1]. Unlike Bryant's approach, our algorithms accept OBDDs with complement edges, and construct a reduced OBDD in only one step. In addition, our algorithms are much faster than those presented by Bryant and need less memory. To show the efficiency of our method, the results were compared to those obtained whith the BDD software package, described in [6].

It is also shown how the proposed ROBDD generations for combinational circuits can be extended to sequential circuits. For sequential circuits, the ROBDD generation is based on a modeling technique that transforms a synchronous sequential circuit into an

iterative combinational array. Since, generally, a sequential circuit starts from a totally unknown state, time is taken into account.

The paper is organized as follows: in Section 2.2, the fundamental principles of ROBDD are reviewed. In Section 2.3, our new boolean function manipulation algorithms based on ROBDDs are presented. Section 2.4 shows some experimental results obtained for combinational circuits. Section 2.5 deals with ROBDD generation for sequential circuits. In Section 2.6, we present the experimental results obtained for ISCAS'89 benchmark circuits. Finally, conclusion is given in Section 2.7.

2.2 Reduced Ordered Binary Decision Diagram Review

The fundamental principles of ROBDD are reviewed in this section.

Given a boolean function F and an ordering of decision variables $(x_1, x_2, \ldots, x_n)$, an OBDD is the graph representation of the Shannon expansion of F according to the given variable ordering. For example, the OBDD of the function $F(x_1, x_2, x_3)$ described in the truth table of Figure 2.1(a) is shown in Figure 2.1(b). In the figure, each non-terminal vertex is represented by a circle containing the index of a decision variable, and each terminal vertex is represented by a square containing the function value 0 or 1. Although, the "fully expanded" OBDD requires $2^n - 1$ non-terminal vertices, its graph size can be drastically reduced by using the following two reduction rules.

1) *merging rule*: two isomorphic subgraphs should be merged.

2) *deletion rule*: a vertex whose two branches point to the same vertex should be deleted, since, in this case, the value taken by the function corresponding to the BDD will be independent of the value of the input corresponding to the vertex.

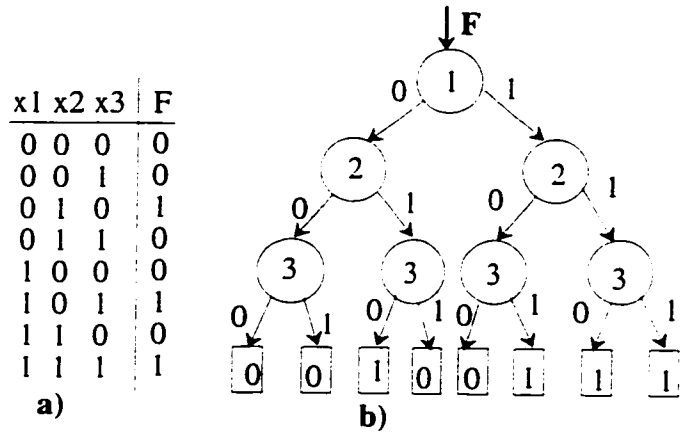


Figure 2.1 A boolean function and its OBDD

2.2.1 Bryant's approach [1]

Bryant [1] presented a set of algorithms for performing a variety of operations on boolean functions. The main procedures are called *Apply*, *Apply_step* and *Reduce*. The procedure *Apply* takes two graphs representing functions f_1 and f_2, a binary operator $\Diamond$ (i.e., any Boolean function of 2 arguments) and produces a reduced graph representing the function $f_1 \Diamond f_2$.

The algorithms are based on the following recursion, which is derived from the Shannon expansion:

$$f_1 \Diamond f_2 = \overline{x}_i \cdot (f_1|_{x_i=0} \Diamond f_2|_{x_i=0}) + x_i \cdot (f_1|_{x_i=1} \Diamond f_2|_{x_i=1}) \qquad (2.1)$$

Bryant obtains a reduced graph in two steps. The Procedure *Apply* calls first the procedure *Apply_step* to generate a graph representing the function $f_1 \Diamond f_2$. Before the final graph is returned, the Procedure *Reduce* is applied to this graph to transform it to a reduced one. This procedure repeatedly applies the two reduction rules, described earlier,

to the generated OBDD until they are no longer applicable. The resultant graph, called the *reduced* OBDD (ROBDD), is unique [1]. For instance, Figure 2.2 represents the ROBDD of the function presented in Figure 2.1(b).

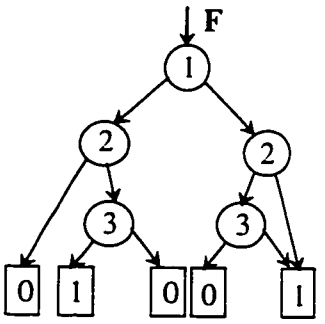


Figure 2.2 ROBDD with only regular edges

To store the result of the algorithm application to a pair of vertices, Bryant uses a table whose size is $|G_1| \cdot |G_2|$, where $|G_1|$ and $|G_2|$ represent the number of nodes in the graphs G_1 and G_2 respectively. G_1 (G_2) represents the BDD corresponding to $f_1(f_2)$.

We have observed that in some trivial cases (the cases where, for instance, one of the nodes is terminal) are not stored in this table, and a very large number of elements in the table are unused. In addition, a time complexity of $O(|G_1| \cdot |G_2|)$ is needed for initializing the elements of this table.

2.2.2 Utility of Complement Edges

Now, suppose that we are interested in generating BDDs for two complement functions G and $\overline{G}$. Using Bryant's approach, the ROBDD nodes for G and $\overline{G}$ are similar except that their terminal nodes are interchanged. This similarity can be exploited by

using complement edges. Therefore, $\overline{G}$ could be represented by a complement edge to the node for G, saving intermediate nodes.

To maintain a canonical form, we must constrain where complement edges are to be used. Karpulus [4], Mardre *et al.* [5] and Karl *et al.* [6] formulated a set of rules to guarantee canonical ROBDDs using complement edges. Our implementation uses the following rule which was described in [6]: The high edge of every node must be a regular edge. Thus, we always choose the left member of each pair of functions as shown in Figure 2.3. Note that these 4 pairs are functionally equivalent. A dot on an edge is used to indicate that it is a complement edge.

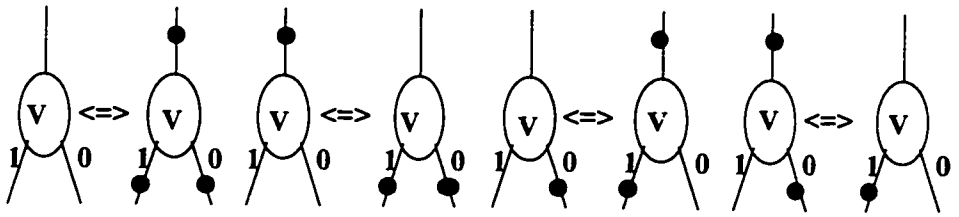


Figure 2.3 BDDs of equivalent functions

Using complement edges with the constraint described above, the size (number of nodes) of the graph of Figure 2.2 will be reduced further. The ROBDD graph of Figure 2.4 represents the same boolean function F of Figure 2.1, with 4 nodes less than the graph shown in Figure 2.1.

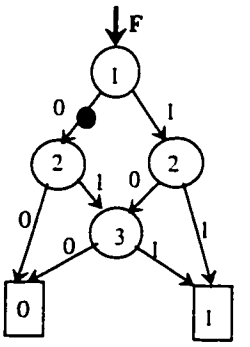


Figure 2.4 Reduced OBDD with the two kind of edges

2.3 Our approach

Our function manipulation algorithms are based on ROBDD representation, like the ones described in [1]. Some modifications have been made to Bryant's algorithms in order to accelerate them and make them accept BDDs containing complement edges. The C programming language code of the data structure, used for ROBDD nodes, is shown in Figure 2.5.

```
typedef struct {
  int low, high;
  unsigned short index, time;
  unsigned short mark; /** visited => mark = 1 else mark = 0 ***/
  short ref; /** the number of times this node is referred to **/
  unsigned short id; /* used to number the ROBDD nodes */
} bdd_vertex, *bdd_vertex_ptr;
```

Figure 2.5 Data structure of a BDD node

One of the characteristics of C programming language is that a vertex is always stored in an address which is a multiple of 4. In other words, the least two significant bits of the

address are always equal to 0. In addition, C language gives the possibility to pass from a pointer to an integer and vice-versa without any problem.

Suppose that v1 is a pointer to a BDD vertex. *I = (int) v1*, represents the address of the vertex referred by *v1*. On the other hand, to find *v1*, we have only to write the following line of C language code: *v1 = (bdd_vertex_ptr) I*.

2.3.1 Complement edge representation

In this section, we will introduce the approach, we have adopted, to represent complement edges. A boolean function is represented by an integer that holds both the information about the top vertex of the corresponding ROBDD and the type of edge connected to this vertex. This way, no memory overhead is added.

Figure 2.5 shows the data structure we use to represent the BDD nodes. According to this figure, we note that *low* and *high* fields of the data structure are declared as integers. These two integers will hold the BDD vertices that are connected to the *high* and *low* edges of the considered vertex. Furthermore, the integer can also be used to refer to a function or to its complement.

Since, using C programming language, the least bit of the storage address of any vertex is always 0. So, it can be used to indicate the kind of edge connected to the vertex. In our approach, the least significant bit is made equal to 1 to indicate that a complement edge is connected to the vertex. In the opposite case, a regular edge is connected to the vertex. Furthermore, the address of a vertex and the type of an edge can be used as a unique identification of any function. In this way, a function and its complement will have

different unique identifications, and two equivalent functions will have always the same identification. Hence, equivalent or complement functions will be easily detected.

In the developed program, the variable *least_bit_0* is defined as 0xFFFFFFFE (all the 32 bits of this variable are equal to 1, except the least significant one). Given a boolean function *F*, represented by an integer *I*, it only takes a binary AND operation between *I* and *least_bit_0* to obtain exactly the address of the vertex corresponding to the function, whatever the kind of edge connected to the vertex is.

Let us take, for example, the following three lines of our code:

```
v = (bdd_vertex_ptr) (I & least_bit_0); /* &: binary AND operation in C language */
e = I & 01; /* e is equal to 1 only when a complement edge is connected to v */
I2 = I^1; /* ^: represents the binary XOR operation in C language */
```

These lines allow to determine the address of the top vertex of the ROBDD corresponding to a function *F*, the kind of edge connected to this ROBDD, and the integer representing the complement of *F*. Here, *bdd_vertex_ptr* represents a pointer to the data structure of a BDD node illustrated in Figure 2.5.

Using our approach, the complement operation and the identification of complement functions take a negligible time. Since the integer **1** has all its bits equal to **0**, except the least one, the complement of *F* represented by an integer *I* is $\overline{F}$ (represented by an integer ***I^1***), where ^ is the binary **XOR** operation in C. In this case, all the bits of *I* remain unchanged except the least significant bit which will be complemented. Note that ***I & least_bit_0*** is always equal to ***(I^1) & least_bit_0***; which confirms the fact that two complement functions always refer to the same ROBDD by different edges. To identify com-

plement functions, only a binary **XOR** between the two corresponding integers is needed. The result of this operation is 1 only when we have two complement functions.

2.3.2 Functional manipulation algorithms

In order to accelerate bdd generation, complement edges are exploited as described in Table 2.1.

Table 2.1 Trivial cases for AND, OR, and XOR operations

AND	0	1	$\bar{f}$	f
0	0	0	0	0
1	0	1	$\bar{f}$	f
$\bar{f}$	0	$\bar{f}$	$\bar{f}$	0
f	0	f	0	f

OR	0	1	$\bar{f}$	f
0	0	1	$\bar{f}$	f
1	1	1	1	1
$\bar{f}$	$\bar{f}$	1	$\bar{f}$	1
f	f	1	1	f

XOR	0	1	$\bar{f}$	f
0	0	1	$\bar{f}$	f
1	1	0	f	$\bar{f}$
$\bar{f}$	$\bar{f}$	f	0	1
f	f	$\bar{f}$	1	0

Note that NAND, NOR and XNOR operations are not considered in Table 2.1. For example, the result of a NAND operation is obtained by computing the result of the AND, and complementing it, which takes a negligible time. The same strategy will be used for finding the result of a NOR operation or XNOR operation.

Since the *high* edge of any ROBDD node is restricted to be a regular one, a boolean function F may refer to a BDD by a complement edge. When, for example, $F = \overline{A}$ where A is an input variable, the BDD generated will represent the complement of F. Then, this BDD will be referred by a complement edge in order to represent F.

The procedure ROBDD_GEN, shown in Figure 2.6, takes as arguments two integers, representing two functions f_1 and f_2, and a binary operator $\Diamond$. It produces a reduced graph representing the function $f_1 \Diamond f_2$. Each integer contains the address of the top vertex of the

corresponding ROBDD and the type of edge. Based on the Shannon expansion the resulting graph is easily built.

According to Equation 2.1, in order to find the result of the binary operation, we have to compute $f1|_{x_i=0}$, $f1|_{x_i=1}$, $f2|_{x_i=1}$ and $f2|_{x_i=1}$. Since we use complement edges, we have to consider two cases. First, the case when the least significant bit of the integer corresponding to *f1* is 0. In this case, the function refers to its ROBDD by a regular edge. Let *V* be the root node of this ROBDD and *i* be the index of *V*. In this case, *V->low* represents $f1|_{x_i=0}$ and *V->high* represents $f1|_{x_i=1}$. In the second case, when the least significant bit is equal to 1, a complement edge is referring to the ROBDD. In this case, *V->low* and *V->high* represent $\overline{f_1}\big|_{x_i=0}$ and $\overline{f_1}\big|_{x_i=1}$, respectively. In other words, *(V->low ^ 1)* and *(V->high ^ 1)* represent $f1|_{x_i=0}$ and $f1|_{x_i=1}$, respectively. The same approach will be used to find $f2|_{x_i=0}$ and $f2|_{x_i=1}$, as illustrated in Figure 2.8.

To apply the operator to two functions f_1 and f_2 (Figure 2.8), we must consider several cases. First, let us suppose trivial cases, such as the case when one of the two functions is constant, or the case where f_1 and f_2 are either equivalent or complement functions. For these cases, our procedure, called *terminal_case*, shown in Figure 2.7, computes the result as shown in Table 2.1. Otherwise, suppose that *index(v1)* = *index(v2)* = *i*, a vertex *u* containing index *i* is created, and the procedure CONST_BDD (Figure 2.8), is applied recursively on *low(v1)* and *low(v2)* to generate the subgraph whose root becomes *low(u)*. Similarly, CONST_BDD is applied recursively on *high(v1)* and *high(v2)* to generate the subgraph whose root becomes *high(u)*. Suppose, on the other hand, that *index(v1)* = *i*, but either *v2* is a terminal vertex or *index(v2)* > *i*. Then, the func-

tion, represented by a graph whose root node is *v2* is independent of the logic value taken by x_i, i.e.,

$$f_2|_{x_i = 0} = f_2|_{x_i = 1} = f_2 \tag{2.2}$$

Hence, we create a vertex having index *i*, but recursively apply CONST_BDD on *low(v1)* and *v2* to generate the subgraph whose root becomes *low(u)*, and on *high(v1)* and *v2* to generate the subgraph whose root becomes *high(u)*. A similar situation holds when the roles of the two vertices in the previous case are reversed.

The algorithms used for ROBDD generation are presented in Figure 2.6 and Figure 2.7. First, the algorithm does not need to evaluate a given pair of subgraphs more than once. Instead, we maintain a table, called *bdd_table*, containing entries of the form *(v1,v2,u)* indicating that the result of applying the algorithm to two integers *v1* and *v2*, was an integer *u*. These integers represent, at the same time, subgraph addresses and the kind of edges.

It is interesting to note that we do not need to store trivial cases in *bdd_table* (Table 2.1), since the result can be easily computed. Note that, before applying CONST_-BDD, the procedure *find_result_of(v1,v2)* (Figure 2.8) is called to check whether *bdd_-table* contains an entry for these two integers. If so, we immediately return the result. Otherwise, we compute *u->low* and *u->high*, and store *u* in bdd_table.

In our approach, we maintain a table, called ROBDDs_TABLE, that contains all the generated BDDs. In this table, we do not find any two different BDDs that represent the same boolean function.

To have a reduced OBDD, we always compare *high(u)* with *low(u)*. If they are equal, then the two branches of u point to the same vertex with the same kind of edge. As a result, u will be deleted, since the value taken by the corresponding input variable will have no influence on the result, and u->*low* will be returned and its reference count is decremented. In the opposite case, u->*low* $\neq$ u->*high*, a lookup in ROBDDs_TABLE, with u->*low* and u->*high* used as keys, determines whether a node equivalent to u already exists. If a node equivalent to u is found, the existing node is returned and u is freed. Note that, unlike Bryant's approach, which requires two steps to compute a reduced and ordered BDD, our approach leads to a reduced graph in just one step .

```
bdd_function ROBDD_GEN(f1, f2, binary_operation)
    int f1, f2, binary_operation;
 {
    n1 = number_nodes(f1); /* gives a number to each vertex of the BDD
                              and returns the total number of vertices */
    n2 = number_nodes(f2);
    if(n1 > n2)
      n1 = number_nodes(f1);
    else {
      f = f1;
      f1 = f2;
      f2 = f;
      n1 = n2;
    }

    /* Here, the BDD of f1 has always more nodes than that of f2 */
    bdd_table = (bdd_table_ptr *) malloc(sizeof(bdd_table_ptr)*n1);
    for (i = 0; i < n1; i++)
      bdd_table[i] = NULL;  /* Initialize the bdd_table */
    u = const_ROBDD(f1, f2, binary_operation);
    for (i = 0; i < n1; i++)
      free (bdd_table[i]);
    free (bdd_table);
    return(u);
 }
```

Figure 2.6 Procedure ROBDD_GEN

As indicated in Figure 2.6, *bdd_table* is a table of chained lists whose size is $MAX\{|G_1|, |G_2|\}$. The nodes corresponding to the function having the biggest size are numbered, a number is stored at the field *id* of the node. *bdd_table*[*i*] corresponds to the entries *(v1,v2)*, where *i represents the identification of v1 (v1->id)*. *v1* belongs to the OBDD of the biggest size. Since, in our approach, the trivial cases are not stored, the number of elements to be stored in *bdd_table* is drastically reduced. Note that, when each list contains a constant number of elements, a lookup in *bdd_table* or ROBDDs_table, will

take a constant time. As a result, the complexity of the procedure CONST_ROBDD, Figure 2.8, will be linear.

```
bdd_function terminal_case(v1, v2, binary_operation)
      int v1, v2, binary_operation
 {
   switch(binary_operation) {
     case AND:
       if(v1 == zero || v2 == zero) return (zero);
       if(v1 == one) return(v2);
       if(v2 == one) return(v1);
       if(v1 == v2)  return (v1);
       if((v1 ^ v2) == 1) return (zero);
     break;
     case OR:
     if(v1 == one || v2 == one) return (one);
     if(v1 == zero)  return(v2);
     if(v2 == zero ) return(v1);
     if(v1 == v2)  return (v1);
     if((v1 ^ v2) == 1) return (one);
     break;
     case XOR:
     if(v1 == zero)  return(v2);
     if(v2 == zero)  return(v1);
     if(v1 == one)   return(v2 ^ 1);
     if(v2 == one)   return(v1 ^ 1);
     if(v1 == v2)    return(zero);
     if((v1 ^ v2) == 1)  return(one);
     break;
   }
   return (find_in_BDD_TABLE(v1,v2));
}
```

Figure 2.7 Procedure terminal_case

To study complex functions, we must have an effective memory management scheme. In our implementation, a garbage collection is performed whenever the number of dead nodes exceeds 15% of the total number of nodes.

```
int CONST_ROBDD(v1, v2, binary_operation)
    int v1, v2, binary_operation;
 {
    vlow1 = terminal_case(v1,v2);
    if(vlow1 != 0) {
      u_vertex = (bdd_vertex_ptr) (vlow1 & last_bit_0);
      u_vertex->ref++; return(vlow1);
    }
    v1_adress = v1 & last_bit_0;
    v1_vertex = (bdd_vertex_ptr) v1_adress;
    v2_adress = v2 & last_bit_0;
    v2_vertex = (bdd_vertex_ptr) v2_adress;
    min_index = MIN(v1->index,v2->index);
    if(v1->index == min_index) {
    if(v1 == v1_adress){/* In this case, the ROBDD is referred by a regular edge */
        vlow1 = v1_vertex->low;
        vhigh1 = v1_vertex->high;
      }
      else {/* In this case, the ROBDD is referred by a complement edge */
        vlow1 = v1_vertex->low ^ 1;
        vhigh1 = v1_vertex->high ^ 1;
      }
    }
    else {
      vlow1 = v1;
      vhigh1 = v1;
    }
    if(v2_vertex->index == min_index) {
      if(v2 == v2_adress) {
        vlow2 = v2_vertex->low;
        vhigh2 = v2_vertex->high;
      }
      else {
        vlow2 = v2_vertex->low ^ 1;
        vhigh2 = v2_vertex->high ^ 1;
      }
    }
    else {
      vlow2 = v2;
      vhigh2 = v2;
    }
    vlow1 = CONST_ROBDD(vlow1, vlow2, binary_operation);
    vhigh1 = CONST_ROBDD(vhigh1, vhigh2, binary_operation);
    if (vlow1 != vhigh1)
     vlow1= find_equivalent_or_insert_in_ROBDD_table(min_index,vlow1,vhigh1);
    else {
      ((bdd_vertex_ptr) (vlow1 & last_bit_0))->ref--; /* deletion rule */
     insert_bdd_table(v1,v2,vlow1);
     return(vlow1);
    }
}
```

Figure 2.8 Procedure CONST_ROBDD

As shown in Figure 2.5, the reference count, corresponding to a given node N, which is the field *ref* of the data structure, contains the number of other nodes that refers to it. N is called dead, when its reference count is 0, which means that it is not referred by any BDD node. So, it can be freed without creating any problem. In addition, when a user defined function is no longer needed, the reference count of the corresponding node is decremented. If the new reference of N is 0, then the reference counts of the nodes connected to its branches *low* and *high* will be recursively decremented.

2.4 Experimental Results

In this section, we present the run-time and memory requirements for converting the combinational portion of several digital circuits into the ROBDD representation. These circuits are standard benchmarks [7] available in the public domain and they are known to have large sum-of-products representations.

The results shown in Table 2.2 are obtained with a fixed order for all the primary inputs. Table 2.2 also compares our results with the results obtained when the BDD Package is applied [6]. Both algorithms have been run on the same station with the same variable ordering.

Table 2.2 ROBDD results for ISCAS'85 benchmark circuits

Circuit	#In	#Out	Our Approach		BDD Package [6]	
			Memory (Mb.)	CPU (sec.)	Memory (Mb.)	CPU (sec.)
c432	36	7	1.27	1.41	0.8	2.3
c499	41	32	1.28	18.21	1.3	19
c880	60	26	1	3.62	0.8	5.3

Table 2.2 ROBDD results for ISCAS'85 benchmark circuits

Circuit	#In	#Out	Our Approach		BDD Package [6]	
			Memory (Mb.)	CPU (sec.)	Memory (Mb.)	CPU (sec.)
c1355	41	32	2.7	43.22	3	156.7
c1908	33	25	1.2	20.41	1.2	19.2
c3540	50	22	5.4	77.31	5.3	87
c5315	178	123	1.6	7.86	1.8	14.86

In Table 2.2, we have the circuits with ROBDDs generated for all the primary outputs with a single ordering. In the first column of the table, the name of the circuit is followed by the name of its primary inputs and outputs. Column 4 gives the memory used by our approach. The algorithm was run on a Sun SPARC Station 2. Column 5 gives the CPU time, in seconds, used by our algorithm. Columns 6 and 7 gives the results obtained when the BDD package described in [6] is used.

According to this table, we note that in most cases our approach is faster than that presented in [6]. The memory required by both algorithms is almost the same. Note that for circuit c1355, our algorithm is three times faster.

2.5 BDD Generation for Sequential Circuits

In this section, we will show how our BDD generation approach for combinational circuits can be extended to sequential circuits.

For sequential ROBDDs construction, we have to take into account the fact that, generally, a sequential circuit starts from a totally unknown state. In addition, the present state of a sequential circuit depends on its previous states.Therefore, the modeling should take into consideration these two observations.

We have found that the iterative array model is appropriate for sequential circuit modeling. This modeling technique maps the time domain response of a sequential circuit into a space domain response of the iterative array. Since this latter is a combinational circuit, BDD generation will be possible for sequential circuits.

The maximum length of the iterative model has to be chosen properly. It can be either specified by the user or estimated, using probabilistic testability measures (TMs). These measures can be used also for ordering the inputs of the iterative model.

2.5.1 Sequential Circuit modeling

In this paper, we restrict ourselves to synchronous sequential circuits whose components are gates and clocked flip-flops (F/Fs). The method is based on the modeling technique that transforms a synchronous sequential circuit into an iterative combinational array (Figure 2.9). One cell of this array is called a *time-frame*. In this transformation, a flip-flop is modeled as a combinational element having an additional input y to represent its current state and an additional output y^+ to represent its next state, which becomes the current state in the next *time-frame*. An input vector of the combinational array represents an input sequence for the sequential circuit.

Because all the cells in the iterative array have an identical structure, there is no need to actually construct the complete model of the iterative array, and the same structural model can be used for every *time-frame*. Thus, for BDD generation, a synchronous sequential circuit S can be modeled by a pseudo-combinational iterative array, as shown in Figure 2.9. In this model, each cell $C(i)$ of the array is a combinational circuit. If an input sequence $x(0), x(1),..., x(k)$ is applied to S, which is in an initial state $y(0)$, S gener-

ates the output sequence *z(0), z(1),..., z(k)* and states *y(1), y(2), ..., y(k)* (for sequential nodes).

The iterative array will generate the output *z(i)* from cell *i*, in response to the input *x(i)* to cell *i* ($1 \leq i \leq k$). Note that the first cell also receives the values corresponding to *y(0)* as inputs. In this transformation, the clocked F/Fs are modeled as combinational elements, referred to as pseudo-F/Fs. The present state *y* of the F/Fs in cell *i* must be equal to the y^+ output of the F/Fs in cell *i*-1.

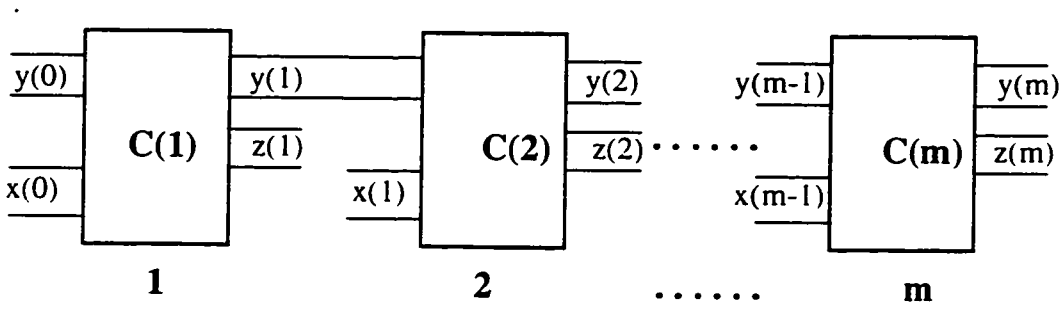


Figure 2.9 Iterative array model of a sequential circuit [52].

An important question is when to stop increasing the number of time-frames? Sequential Testability Measures will answer this question.

2.5.2 Sequential TMs

In this section, the probabilistic TMs for sequential circuits and their use in sequence length estimation are reviewed.

Naim B. et *al.*, [9] and [10], used the *Initializability* concept for sequential circuits testability evaluation. Initializability is a quantification of the difficulty in controlling a sequential node given that it starts from an unknown state. With the initializability con-

cept, TMs can be computed for sequential modules. The initializability measure is a probabilistic measure whose computation is similar to COP number computation [8].

Soufi et *al.* [11] described sequential controllabilities that are computed as a function of time, using an iterative array model. These measures had been used to enhance the testability of sequential circuits and to make them fully testable using only pseudo-random test vectors. Based on Markov chain theory, it had been proven that these measures are non-decreasing functions of time. This characteristic is used by our algorithm for computing the maximum number of time frames to be considered.

According to [11], we distinguish three different controllabilities, denoted by $C_0(h, t)$, $C_1(h, t)$ and, $C_X(h, t)$, which represent the probabilities that h takes the logic value 0, 1 or X (unknown), respectively, after applying t random vectors. Since the sum of these three terms is always equal to one, only two of the terms are to be computed; the third can be deduced from the others.

Here, only C_0 and C_1 will be computed [11]. Initially (at time 0), the circuit is supposed to be in an unknown state. In this case, the controllabilities $C_0(h, 0)$ and $C_1(h, 0)$ of all the flip-flops are set to 0 (which means that $C_X(h, 0) = 1$). However, since the primary inputs can be accessed at any time t and the vectors applied are randomly chosen, then their values can be set to 0 or to 1, with an equal probability. Then, at each time frame, C_0 and C_1 of a primary input are made always equal to 0.5. For the other lines of the circuit, $C_0(h, t)$ and $C_1(h, t)$ are computed using the formulas, given in [11], which depend on the type of gate.

In the other hand, for a flip-flop F, at any time t ($t>0$), $C_0(F,t) = C_0(I,t-1)$ and $C_1(F,t) = C_1(I,t-1)$, where I is the input of the flip-flop and F is its output. In other words, the controllabilities of the flip-flop output at time $t+1$ will be those computed for the input line of this flip-flop at time t.

The formulas for the computation of sequential TMs are presented in [11]. Since the sequential controllabilities are non-decreasing functions of time, they are used as a stopping criterion in the iterative model. In our case, a controllability is considered stable whenever the difference between this controllability at time t and that at time t-1 is less than ε (In this paper, ε is chosen to be equal to 10^{-4}). When the difference is less than ε in a line, it means that, by applying any sequence of $t+1$ vectors, v_1 to v_{t+1}, the vector v_1 will have no effect (with a probability close to 1) on the logic value taken by the line. This also means that, even if v_1 is removed from this sequence, the last value taken by the line will be the same, with a high probability.

The maximum length of a test vector sequence is the number of circuit duplications for which all the controllabilities become stable.

2.5.3 ROBDD Generation Procedures

We have found that BDD generation can be extended to sequential circuits, using an iterative array model. We have found also that the functionality of any line of a sequential circuit can be represented by a BDD, provided that each BDD node contains the time as well as the index of the corresponding input variable.

Using an iterative array model, the complexity of the circuit considered increases with the number of duplications, since the number of nets in the iterative array model is

equal to the original number of nets of the sequential circuit multiplied by the number of time-frames (duplications) considered. As a result, BDD sizes will be also increased. Then, in order to have manageable BDDs, the number of duplications has to be limited, and a good variable ordering has to be found.

In our approach, the maximum number of duplications to consider is obtained using probabilistic testability measures. While computing the sequential controllabilities, we only have to find the time t when all the controllabilities become stable.

2.5.4 Variable Ordering

The BDD sizes depend greatly on variable ordering. Here, we will show how the probalistic testability measures introduced in Section 2.5.2 can be used for variable ordering.

First, in order to have good ordering for the entire circuit, the inputs which are close to the primary outputs must be ordered before the others. In other words, according to the iterative array model, an input A at time t (or $A[t]$) must be ordered before the same input at time t-1 ($A[t\text{-}1]$). However, to order the input variables of the same time-frame, we use a strategy based on sequential depths of the nets. We mean by sequential depth of a net the time when all the controllabilities of the net become stable.

Before ordering the input variables, our algorithm orders first the inputs of each gate in the sequential circuit. For each gate, the input with the lowest sequential depth is ordered first. This way, while ordering the input variables, the net connected to this input will be traversed first. The circuit is traversed from an output to the primary inputs in a depth-first-search manner. The first primary input reached is made first in the BDD order-

ing, the second primary input reached is ordered second, and so on until all the primary inputs that can be reached from the output are ordered.

Since, generally, it is not always possible to find a single good ordering for all the primary inputs of a circuit, the primary outputs (POs) of the sequential circuit can be studied separately. The main advantage of this approach is that it enables us to study much large circuits. The only disadvantage is that the CPU time used may be greater than the corresponding time in the case where a single good ordering can be found easily. Since the BDDs will be generated for many time frames, the number of variables to consider will increase. Consequently, it is very convenient to study the primary outputs separately.

2.5.5 ROBDD generation procedures for sequential circuits

In this section, we will present the approach we use for generating ROBDDs for sequential circuits. Since the initial state of a sequential circuit is generally unknown, we have to add another terminal node that represents the logic value X.

Here, we will show how to modify our BDD generation method for combinational circuit in order to manipulate boolean functions with unknown states. Furthermore, to represent the behavior of any net of the sequential circuit, time is included in the data structure of each BDD node.

The procedure, shown in Figure 2.10, is used to generate BDDs for a primary output and all the lines that can be reached by this output. After selecting a primary output, the input variables of the sequential circuit are ordered according to the sequential depths of the circuit nodes, as illustrated in Figure 2.10. In the beginning of BDD generation, we assume that the circuit starts from a totally unknown state. In other words, the BDDs cor-

responding to the flip flops are made equal to terminal node X. Note that the BDD corresponding to the output of a flip-flop at time frame t is made equal to the BDD computed for its input at time frame t-1.

```
Build_BDD_of(PO)
  int PO
{
    - Order the input variables
    for(j = 0; j < number_of_FFs; j++)
        BDD[FF[j]] = X;
    for(t=0; t < max_t; t++){
        generate_bdd(PO);
        for(j =0; j < number_of_FFs; j++)
            tmp_BDD[FF[j]] = generate_bdd(FF[j]);
        for(j =0; j < number_of_FFs; j++)
            BDD[FF[j]] = tmp_BDD[FF[j]];
    }
}
```

Figure 2.10 BDD construction for a primary output and all the lines corresponding to its circuit

In order to have reduced BDDs, we have to take into account the fact that, when a variable is unknown, its complement is also unknown. In other words, when $F=X$, $\overline{F}=X$. The function and its complement will refer to the same terminal node with the same kind of edge. This does not mean that F is equal to $\overline{F}$. It is interesting to note that, when this characteristic is not taken into consideration, the BDDs generated may not be compacted. Let us take, for example, the circuit shown in Figure 2.11. Suppose that we are interested in finding the BDD corresponding to the primary output of this circuit.

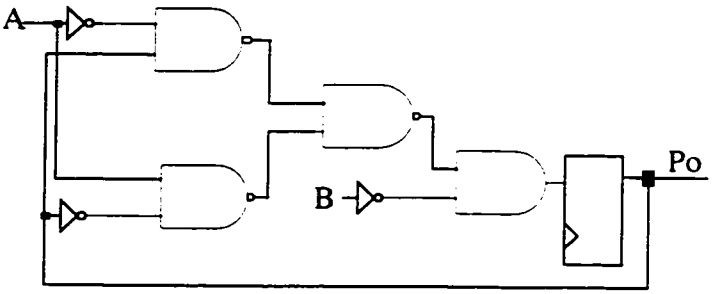


Figure 2.11 A Sequential Circuit

According to Figure 2.12a, we note that the high and low edges refer to the same terminal node X with two different edges. In this case, the deletion rule (Section 2.2) cannot be applied. But, when taking into account the fact that the complement of an unknown value is also unknown, the BDD can be reduced, as shown in Figure 2.12b.

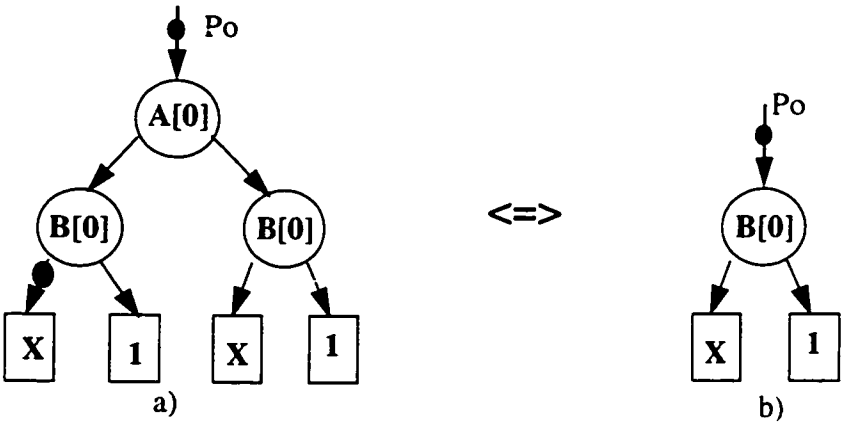


Figure 2.12 BDD computed for line L at time 0

Since the high edge of any BDD node is restricted to be always regular, it might happen that the procedure *terminal_case* (Figure 2.13) receives as argument the terminal node X, referred by a complement edge. Note that, when the result is unknown, this procedure always returns terminal node X referred by a regular edge.

When two functions refer to the same BDD, it does not mean that they are equal or complement. For a sequential circuit, two functions referring to the same BDD with the same kind of edge are equal only when the corresponding BDD does not contain any terminal node X. However, when the BDD contains terminal nodes corresponding to X, and when the functions are supposed equal, depending on the binary operation, the result may be incorrect.

For example, the BDD corresponding to the result of an XOR operation between two functions, referring to the same BDD that contains terminal node X, is not the terminal node 0, but a BDD that contains terminal nodes 0 and X, since $X \oplus X = X$. However, based on the Shannon theorem, we can prove that the result of an AND operation, or an OR operation between two functions referring to the same BDD with the same kind of edge, is a function that refers also to the same BDD. As shown in Figure 2.13, this characteristic is used by the procedure *terminal_case* for accelerating BDD generation.

Now, suppose that we are interested in computing an operation between two functions referring to the same BDD with different edges. Two different cases can occur. The first is the case when the corresponding BDD contains only 0 and 1 terminal nodes. In this case, whatever the operation is, the BDD representing the result will be either the terminal node 0 or the terminal node 1. The second case is when the BDD referred by both functions contains the terminal node X. In this case, whatever the operation is, the BDD representing the result will contain the terminal node X.

```
int terminal_case(v1, v2)
     int v1, v2;
{
     switch(binary_operation) {
     case AND:
          if(v1 == node_zero || v2 == node_zero) return (node_zero);
          if(v1 == node_one){
          if(v2 == node_X_not) return(node_X)
          return(v2);
     if(v2 == node_one){
     if(v1 == node_X_not)  return(node_X);
     return(v1);
     if(v1 == node_X || v1 == node_X_not)
          if(v2 == node_X || v2 == node_X_not)
               return(node_X);
      if(v1 == v2) return (v1);
     break;
     case OR:
     if(v1 == node_one || v2 == node_one) return (node_one);
     if(v1 == node_zero){
          if(v2 == node_X_not) return(node_X);
          return(v2);
     }
     if(v2 == node_zero){
          if(v1== node_X_not) return(node_X);
          return(v1);
     }
      if(v1 == node_X || v1 == node_X_not)
          if(v2 == node_X || v2 == node_X_not)
               return(node_X);
     if(v1 == v2) return (v1);
     break;
     case XOR:
     if(v1== node_X || v1 == node_X_not || v2 == node_X || v2==node_X_not) return(node_X);
     if(v1 == node_zero) return(v2);
     if(v2 == node_zero) return(v1);
     if(v1 == node_one) retun(v2 ^ 1);
     if(v2 == node_one) return(v1 ^ 1);
     break;
     }
     return (find_v1_v2(v1,v2));
}
```

Figure 2.13 Procedure terminal case for sequential

The procedure *terminal_case*, shown *in* Figure 2.7, has been modified, as described in Figure 2.13, in order to take into consideration the fact that, for some assignments to the primary input variables, the logic values of a function are unknown. To have reduced BDDs, *terminal_case* always returns the terminal node X referred by a regular edge whenever the result of a boolean operation is unknown. As shown in the same figure, when two functions refer to the same BDD, they are not considered equal or complement functions.

In order to generate BDDs for sequential circuit, we included time in the data structure of the BDD nodes. In this way, the same input variable can take different logic values at different time frames. Figure 2.14 shows the modifications that were done in the procedure of Figure 2.6, to take time into consideration in BDDs' generation.

When the indices of two BDD nodes are equal, but the times are different, the corresponding variables are supposed to be different. The procedure Min(v1,v2), shown in Figure 2.14, takes as the arguments two nodes and returns the node whose corresponding variable has to be ordered first. Then, if the time corresponding to v1 (v2) is less than that corresponding to v2 (v1), v1 (v2) will be returned, whatever the index of v2 (v1) is.

```
if(v1 == node_zero || v2 == node_zero) return (node_zero);
    u_vertex = MIN(v1_vertex,v2_vertex);
    if(v1_vertex->index == u_vertex->index && v1_vertex->time == u_vertex->time) {
        if(v1 == v1_adress){
            vlow1 = v1_vertex->low;
            vhigh1 = v1_vertex->high;
        }
        else {
            vlow1 = v1_vertex->low ^ 1;
            vhigh1 = v1_vertex->high ^ 1;
        }
    }
    else{
        vlow1 = v1;
        vhigh1 = v1;
    }
    if(v2_vertex->index == u_vertex->index &&    v2_vertex->time == u_vertex->time)   {
        if(v2 == v2_adress) {
            vlow2 = v2_vertex->low;
            vhigh2 = v2_vertex->high;
        else {
            vlow2 = v2_vertex->low ^ 1;
            vhigh2 = v2_vertex->high ^ 1;
        }
    }
    else {
        vlow2 = v2;
        vhigh2 = v2;
    }
```

Figure 2.14 The modifications to be done to take time into consideration

Example 1

The circuit of Figure 2.15a is used to illustrate the ROBDD generation for a sequential circuit starting from an unknown state. It illustrates also the effect of the ordering on BDD size. The ROBDDs of the output *Po* are generated in time-frames 0, 1 and 2, and are

presented in Figure 2.15b, 15c and 15d respectively. The variable *A[0]* represents the input *A* at time-frame 0.

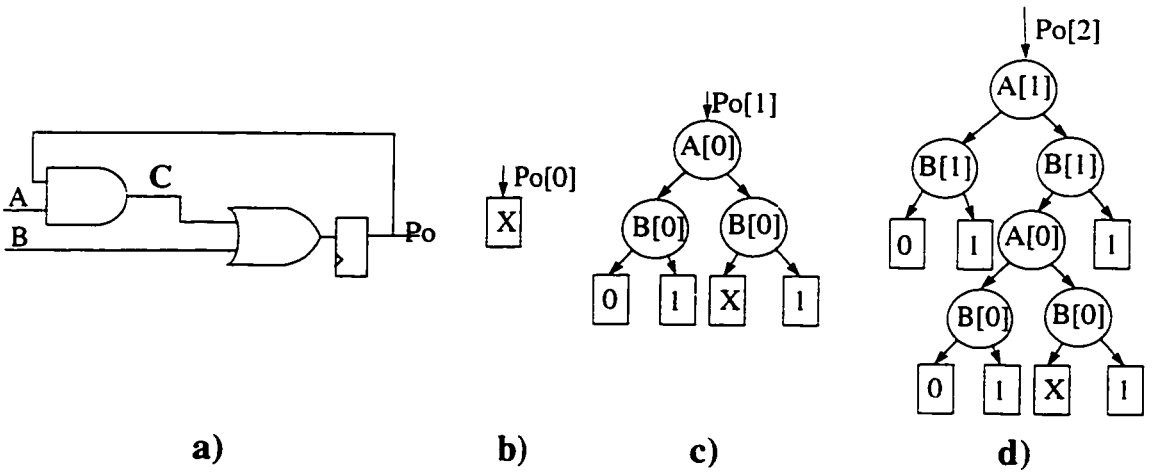


Figure 2.15 ROBDD of Po in time-frame 0, 1 and 2

In *time-frame* 0, the primary output Po, which is also the output of the flip-flop, starts from an unknown state, as indicated in Figure 2.15b. According to Figure 2.15c and Figure 2.15d, after only one time frame, Po can be set to 1 or to 0, since there is a path leading to terminal node 0 and another to terminal node 1. Note that the variables in time-frame 1 (*A[1] and B[1]*) are ordered first, followed by the variables in time 0 (i.e., A[0] and B[0]).

When the sequential depth of each net is considered in variable ordering, BDD sizes can be reduced. For the circuit of Figure 2.15a, the sequential depth of line *B* is less than the sequential depth of line *C*. In Figure 2.15c, the input variable A is ordered before input variable *B* in each time-frame. In Figure 2.16, we show the case when the sequential depth is taken into consideration. As shown in Figure 2.16, by ordering *B* before *A*, the BDD size is reduced.

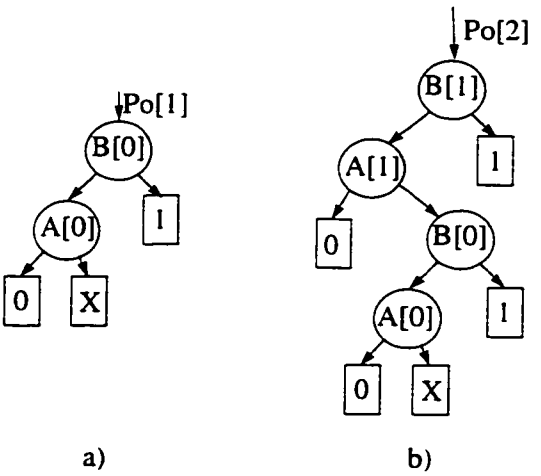


Figure 2.16 ROBDD of Po in time-frame 1 and 2 when the sequential depth is taken into consideration

2.6 Experimental results

In this section, we present the results of BDD generation for some of the ISCAS'89 benchmark circuits [12]. The variable ordering is based on the sequential depth of circuit nets.

In Table 2.3, we show the circuits with ROBDDs generated for all the primary outputs. In the first column of the table we have the name of the circuit followed by the number of primary inputs, the number of primary outputs, and the number of the flip-flops in the circuit. In the other columns and for each circuit, we show the maximum BDD size found in each time frame. We find also the time, in seconds, spent by the algorithm to generate the BDDs for 7 *time-frames*.

Table 2.3 BDD generation for ISCAS'89 benchmark circuits

Circuit	#PIs	#POs	#FFs	Max BDD size (in each time frame)						Time(s)
				1	2	3	4	5	6	
s27	4	1	3	5	14	32	44	56	68	0.2
s420	19	2	16	23	84	157	368	451	675	3.5
s510	19	7	6	1	1	1	1	1	1	2.3
s526	3	6	21	3	5	7	10	14	22	7.4
s641	35	24	19	16	44	1423	2988	78903	159088	119.7
s953	16	23	29	1	4	22	30	52	52	11.3
s1423	17	5	74	4	21	152	1431	18800	43883	97.8
s1494	8	19	6	3	8	23	35	82	157	17.5

According to this table, we note that the sizes of the BDDs to manipulate, generally, increase with the number of time frames. This is due to the fact that the complexity of the array model increases at each time frame. According to the same table, we note that, for some circuits, after a certain number of time-frames, BDD sizes of the sequential nodes remain unchanged. This is due to the fact that the logic values of the sequential nodes may depend only on a certain number of vectors.

For circuit s510, we have found that the BDDs of the primary outputs and flip-flops of this circuit remain equal to terminal node X, even after 100 time frames. Consequently, BDD sizes of the nodes are always equal to 1, as indicated in Table 3. This shows that it is impossible to set any flip-flop to a known value after applying any sequence of 100 vectors. Note that this characteristic of BDDs helps in test vector generation for sequential circuits.

2.7 Conclusion

We have presented a set of efficient and simple algorithms for performing a variety of operations on boolean functions represented by ROBDDs. By combining concepts from Boolean algebra with techniques from graph theory, we have achieved a high degree of efficiency. Our approach uses complement edges to reduce the ROBDDs and the computation time. The experimental results obtained for some ISCAS'85 benchmark circuits show the efficiency of our algorithms.

We have also presented the modifications that are introduced in order to study the sequential circuits. We have shown that, by using the iterative array models of sequential circuits, BDDs can be generated for these circuits. By only computing the BDDs, we can say if a flip-flop can be set to a known state or not after a certain number of vectors. This will be very useful, for many applications such as test vector generation.

Note that this approach cannot be used for generating BDDs for complex sequential circuits, since the length of the iterative array to consider for these circuits is very high. Consequently, the memory and CPU time requirements will be also prohibitive. On the other hand, this approach can be very useful for analyzing and testing finite state machines.

CHAPITRE 3

BDD_FTEST: Générateur de vecteurs de test rapide basé sur le BDD

Introduction

Le présent chapitre est basé sur un papier de conférence intitulé "*BDD_FTEST: Fast, Backtrack-Free Test Generator Based on Binary Decision Diagram Representation*" qui a été publié dans ISCAS en Mai 1995.

Dans ce chapitre, nous décrirons deux approches différentes qui peuvent être utilisées pour générer des vecteurs de test pour un circuit combinatoire. Les deux approches sont des approches algébriques, c'est-à-dire basées sur la manipulation de fonctions logiques. Les fonctions sont représentées par des diagrammes de décisions binaires, pour faciliter leur manipulation. Selon le cas qui se présente, il est plus avantageux d'utiliser l'une ou l'autre des approches.

La première approche, présentée dans cet article, utilise un ordonnancement unique pour toutes les sorties primaires. Elle utilise la notion de dominance pour accélérer la génération de vecteurs de test. Pour générer un vecteur de test, le BDD correspondant au circuit normal (sans la présence de pannes) et celui correspondant au circuit défectueux seront évalués et comparés seulement aux lignes dominateurs [28] (dominators)[1].

Dans la première approche, BDD_FTEST prend une panne et essaie de propager l'erreur résultante à une sortie primaire du circuit considéré. Lorsque les BDD correspondant au circuit normal et celui au circuit défectueux (contenant cette panne), évalués à une des sorties du circuit, sont différents, dans ce cas, un vecteur peut être généré pour tester cette panne. A partir des BDD construits, un vecteur est généré en un temps linéaire. Il s'agit tout simplement de trouver une assignation aux entrées primaires du circuit qui permette d'avoir deux chemins dans les deux BDD qui mèneront à deux noeuds terminaux différents.

Comparée à l'algorithme TSUNAMI [13], qui est, lui aussi, basé sur les BDD et qui utilise un ordonnancement unique, nous avons trouvé que notre première approche passe, en général, moins de temps CPU pour générer des vecteurs de test pour un circuit d'essai donné [7].

Pour certains circuits, il n'existe pas un ordonnancement unique à toutes leurs sorties qui permet de générer des BDD pour toutes les lignes des circuits, par contre, des ordonnancements variables peuvent être trouvés. La deuxième approche que nous présenterons étudie les sorties primaires séparément. En utilisant une telle approche, nous avons pu tester des circuits de plus grande complexité.

Généralement, pour tester une panne, on doit premièrement l'activer[1] et ensuite propager l'erreur résultante à une des sorties primaires. Au lieu d'agir ainsi, notre deuxième approche prend le circuit correspondant à une sortie primaire donnée et vérifie s'il y a une

---

1. Une ligne *i* est dite dominateur de la ligne *j*, si et seulement si tous les chemins de *i* vers la sortie considérée passent par la ligne *j*.

1. faire en sorte que les valeurs logiques de la ligne correspondant à la panne soient différentes dans le circuit normal et le circuit défectueux

panne qu'on peut activer et propager l'erreur résultante à cette sortie. Pour savoir si une panne peut être considérée ou non, il suffit simplement de vérifier si la ligne correspondante à cette panne fait partie du circuit de la sortie primaire considérée.

Comme pour la première approche, pour générer un vecteur de test, le BDD correspondant au circuit normal et celui du circuit défectueux seront évalués et comparés seulement aux lignes dominantes [28]. Si les deux BDD générés à une ligne dominante sont égaux, cette ligne sera stockée dans une liste chaînée qui contient toutes les lignes dominantes auxquelles la panne ne peut se propager. Cette liste sera d'une très grande utilité lors de l'étude des sorties primaires restantes. Notons qu'avant de générer les BDD, BDD_FTEST consulte cette liste pour éviter de refaire le même calcul inutilement plusieurs fois.

Premièrement, lorsque le BDD de la sortie du circuit normal et celui de la même sortie du circuit défectueux seront égaux, l'erreur due à la panne, s'il en existe une, ne peut être propagée d'aucune façon à cette sortie. A ce stade-ci, on ne peut pas dire que cette panne est redondante, il faut attendre jusqu'à ce que l'on étudie toutes les autres sorties primaires. Deuxièmement, si les deux BDD construits sont différents, un vecteur de test sera généré dans un temps linéaire. Les pannes qui restent non-détectées, après l'exécution de BDD_FTEST, sont toutes des pannes redondantes.

La deuxième approche permet d'étudier des circuits plus complexes. Notons que pour des circuits dont un bon ordonnancement unique peut être trouvé, la deuxième approche sera plus lente que la première, vu qu'une panne peut être considérée plus qu'une fois.

BDD_FTEST: Fast, Backtrack-Free Test Generator Based on Binary Decision Diagram Representation

Bechir AYARI and Bozena KAMINSKA

École Polytechnique de Montréal

Department of Electrical and Computer Engineering P.O Box 6079, Station A

Montréal, PQ, H3C 3A7

Abstract

This paper presents a new approach for generating test vectors for combinational circuits. In the approach presented here, the automatic test generator, called the BDD_FTEST, uses an algebraic method to find a set of test vectors for single stuck lines.

To efficiently manipulate Boolean functions, the reduced ordered binary decision diagram (ROBDD) representation is used for algebraic processing. However, the size of BDDs is sensitive to the variable ordering. By using ROBDDs and an ordering strategy based on the network topology for each primary output, we have been able to carry out test vectors for a larger set of networks than existing methods based on BDDs.

To accelerate test vector generation and to eliminate backtracking, we compute the BDD of the function of the faulty circuit and that of the fault-free one at the dominators of the faulty lines.

For all the circuits analyzed, the algorithm is faster than previous algebraic methods. Experimental results demonstrate that for most circuits, our algorithm can generate test vectors for all faults in a very short time, particularly for large circuits like the c7552. BDD_FTEST is also very efficient for hard-to-detect faults and redundant faults.

3.1 Introduction

Test pattern generation can be viewed as a branch-and-bound problem [17]: test pattern generation algorithms usually search through the space by systematically branching and bounding until a test pattern is discovered or the search space is exhausted.

Generally, the complexity of a test vector generation algorithm is measured with respect to the number of backtracks involved, and the test generator shows its worst case with redundant (undetectable) faults which do not have any tests. Then, for redundant faults, the test generator fails to generate a test vector after an exhaustive search, which may involve a considerable amount of backtracking.

To reduce the solution space, PODEM [17] and FAN [18] algorithms identify some necessary assignments using local implications. The contribution of SOCRATES [19] was to find additional necessary assignments that could not be found using local implication. Rajski [16] has presented a test pattern generation algorithm that uses the concept of necessary assignment. A 16-value algebra is employed to reduce or eliminate backtracking in automatic test vector generation. However, there typically exist some "hard" faults that

resist such techniques and lead to unacceptably long test generation times when very high single stuck-at fault coverage is desired.

Other techniques use algebraic methods, principally boolean difference, to determine the entire test set for a fault, [20], [21] and [22]. Early algebraic methods were very slow compared to the enumeration techniques. Recently, however, algebraic methods have been proposed that are fast enough to be useful in generating tests for the hard faults that resist enumeration techniques.

In the Larrabee method [21], the product of sums representation of the boolean difference is first found in terms of the primary inputs and the outputs of the gates in the circuit. This is followed by a search to satisfy assignment of this formula. In [23], Smirat et al. have presented a transitive closure based test generation algorithm. A test is obtained by determining signal values that satisfy a Boolean equation derived from the neural network model of the circuit incorporating necessary conditions for fault activation and path sensitization.

The TSUNAMI algorithm [13], on the other hand, is an algebraic method using BDDs. This algorithm involves a path-oriented search similar to the one used by conventional enumeration algorithms. The method determines the set of input assignments that will propagate a fault through a gate in a path from the fault site to a primary output. This makes it possible to stop the computation as soon as a fault is either propagated to some primary output or it becomes unobservable. The set of tests is defined in terms of primary inputs only. TSUNAMI uses the same ordering for all the outputs. However, the disadvantage is that it may be not possible to find a single good ordering for all the primary outputs. This is the reason why TSUNAMI has not succeeded in generating test vectors for bench-

mark circuits c2670, c3540 and c7552 [7]. In addition, TSUNAMI builds the BDDs for the function of the faulty circuit and the function of the fault-free circuit at the output of each gate on a path from the fault site to a primary output. If the faulty BDD is ever equal to the fault-free BDD at the output of any gate, then the fault may not be propagated through that gate and TSUNAMI backtracks to try a new gate. When the fault is untestable, therefore, TSUNAMI must try all the paths to declare the fault redundant.

In this paper we propose two other algebraic methods, also based on the BDD representation. Our program, called BDD_FTEST, uses a strategy based on the topology of the multi-level network for ordering the input variables. Using BDDs, the time spent on a redundant fault is as great as the time spent on a detectable fault.

In our two methods, we have eliminated backtracking by comparing the BDD of the fault-free circuit and that of the faulty one only at the dominators of the faulty line. Then, when the two computed BDD's are found equal at a dominator line D which is not a primary output, we stop the computation, since at any dominator of D the BDD of the faulty circuit and that of the fault-free one will be equal.

The first of the proposed algorithms uses the same ordering for all the primary outputs (POs). This approach can be used for logic circuits for which we can generate BDD's for all the primary outputs using a single ordering. We have found that this approach is faster for these cases. Since the BDD's for the subcircuits shared by more than one output will be generated only once.

The disadvantage of the first approach is that, in many cases, it may not be possible to find a single good ordering for all the primary outputs in the large examples (c2670, c3540

and c7552). To study these circuits the first approach becomes unpractical and we must change our strategy of testing.

To generate test vectors for large circuits, we propose our second approach. In this approach, BDD_FTEST uses the observability of a line concept, defined in Section 3.3, to identify the set of faults that may be propagated to a given PO. In this approach the primary outputs are studied separately. Each one has its own variable ordering. To accelerate test vector generation, this method takes into account circuits shared by more than one output. The test vector generation ends when all the primary outputs have been studied. The remaining undetected faults are declared redundant, since they cannot be propagated to any PO.

Using the second approach, we have been able to generate test vectors for large circuits in a very short time. As indicated in Table 3.3 of Section 3.5, for the circuit c7552, test vector generation and redundancy identification is completed in only 95 seconds. For the same circuit, we have found that even with a backtracking equal to 100000, the FAN algorithm is not able to identify all the redundant faults.

The paper is organized as follows: Section 3.2 reviews some basic concepts related to BDDs. In Section 3.3, some useful definitions are presented. Section 3.4 introduces the two versions of our test vector generator, BDD_FTEST. We give some experimental results in Section 3.5 and we conclude in Section 3.6.

3.2 A Review of BDDs

We review some BDD-related terms here. These terms have been defined in [1] and are explained now for easy reference in subsequent sections. Consider the decision graph

in Figure 3.1 for the boolean function F = X1.X2 + X3. Each vertex in the graph corresponds to an input variable. To evaluate F at a given input vector $i = (i_1, i_2, i_3)$, we traverse the graph from the root to the leaves in the following manner: At a vertex with index j, we take the 0 (low) branch if i_j is 0 and the 1 (high) branch if i_j is 1. This process continues until we reach a terminal vertex (0 or 1),which is the value of f for that input vector.

Two BDDs with the same variable ordering represent the same function if and only if the two graphs are isomorphic. Using our implementation [24] for generating ROBDD's, this test is very simple to do. We guarantee that the BDDs of equivalent functions will be indicated by the same pointer. This is called a strong canonical form. The test for equivalence is then a constant-time operation. In [1], Bryant has also presented a polynomial time algorithm (in the size of the BDDs) for manipulating boolean functions. In particular, if we have BDDs for the functions F and G and a binary boolean operator $\Diamond$. We may then obtain the BDD for the function $F \Diamond G$ in polynomial time.

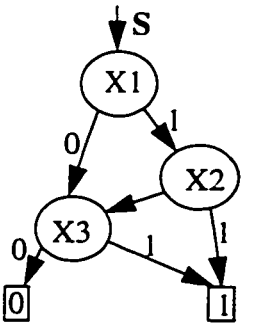


Figure 3.1 An ordered binary decision diagram for the function F = X1X2 +X3

3.3 Basic Definitions

Here we give the definitions used in the subsequent sections. First, the boolean difference is defined as follows:

Definition 1: The boolean difference of any function F with respect to its variable x_i is defined as:

$$\frac{dF}{x_i} = F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \oplus (x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \qquad (3.1)$$

Most algebraic test generation programs use the boolean difference to determine the test set for a fault. In [21], the set of tests for x_i stuck at 0 is obtained by computing $x_i \frac{\partial F}{\partial x_i}$ and the set of tests for x_i stuck at 1 is obtained by computing $\overline{x_i} \frac{\partial F}{x_i}$. TSUNAMI [13] uses a slightly different formula. The test set for the fault α is the set of inputs for which the output of the faulty circuit is different from that of the fault-free circuit. If F_α is the function implemented by the faulty circuit and F the function implemented by the fault-free circuit, then the set of inputs for which F_α is not equal to F is found by satisfying the formula $F \oplus F_\alpha = 1$. As a result, if $F = F_\alpha$ ($F \oplus F_\alpha = 0$), then no test exists and the fault is redundant.

In our implementation, for each undetected fault, we do not need to compute all the set, since this operation will take a polynomial time. Instead, we generate only one test vector. Then we have to find an assignment to the input variables so that the boolean difference will be equal to 1. In other word, the BDD's corresponding respectively to the fault-free circuit and the faulty one will lead to different terminal nodes.

Our algorithm constructs and compares the BDD of the faulty circuit and that of the fault-free one at the dominators of the faulty line. The dominator is defined as follows:

Definition 2: Consider a single-output circuit. Line L is the *dominator* of line l, if all paths from line l to the primary output traverse through the line L.

$$dominator(l) = L \tag{3.2}$$

In this paper, the immediate dominator of a line l means the first dominator line reached by traversing the circuit from line l to the primary output in question. Given a primary output, to find dominators of each line, only the circuit of this primary output is considered.

In the second approach, given a primary output P, we need to compute the observability of each line in the circuit. Given a fault F, BDD_FTEST uses the observability of its faulty line to tell if this fault can be propagated to P or not. The observability of a line is defined as follows:

Definition 3: A line L is observable by the i^{th} primary output (PO) if there exists at least one path from line L to this PO. Then its observability is set to 1, as indicated in Equation 3.3. In the opposite case, the observability of the line is 0.

$$observability[L][i] = 1 \tag{3.3}$$

Given a PO of a circuit C, we obtain the observability of each line in the circuit in two steps. In the first step, the observability of each line of the circuit is made equal to 0. In the second step, we traverse the circuit from this PO to the primary inputs. The observability of each line traversed is set to 1.

3.4 Test Vector Generation

In this section, we present two algebraic methods for generating test vectors to combinational circuits. Both methods are based on BDD representation. We use the heuristic presented in [3], which is based on the topology of a circuit, to order the input variables.

We avoid backtracking by comparing the BDD's only at the dominators of the faulty line. In our two approaches, given a fault F, we begin first by computing the BDD of the faulty circuit and that of the fault-free one at the immediate dominator of the faulty line, let this line be D. If the computed BDD's are different at this line, we continue by computing the BDD's at the dominator of line D. We proceed this way while the BDD's are different. As soon as the BDD's become equal, we stop the computation. When the computed BDD's are different at a primary output, to generate a test vector, we do not need to compute the boolean difference at this output, this operation will take a polynomial time. Instead, we choose an assignment to the input variables so that both BDD's will lead to different terminal nodes. This way, a test vector is obtained in a linear time.

The first approach uses the same ordering for all the primary outputs. This method is proven to be faster when a single good ordering can be found easily for the input variables.

To generate test vectors for large circuits, we propose the second method to be used. This method studies each primary output separately. The variable ordering is obtained by applying the ordering heuristic described in [3] to order the primary inputs.

3.4.1 Test vector generation using the same ordering for all POs

In our first approach we consider a single ordering for all POs. The advantage of using the same ordering for all the primary outputs is that the BDDs for the intermediate nodes

need not be recomputed for each primary output. In addition, an undetected fault is studied only once. This accelerates test vector generation for small circuits, as shown in Table 3.2 of Section 3.5.

To generate the set of test vectors, TSUNAMI computes the boolean difference of the function of the fault-free circuit and that of the faulty one at the output of each gate on a path from the fault site to a primary output. This method is not efficient when the fault is redundant and the boolean difference is 0 only in the dominators of the faulty line. In this case, TSUNAMI computes the boolean difference at many lines in addition to computing the boolean difference at the dominator lines of the faulty line.

Suppose that we have to produce a test vector to an undetected fault α. Suppose, too, that line L is a dominator of the faulty line corresponding to α. To generate a test vector for α, we must compute F_L and F_{L_α} at line L. F_L is the boolean function of the fault-free circuit and F_{L_α} is the Boolean function of the circuit with the presence fault α. Both functions are computed at line L. Then, if L is a primary output, $H = F_L \oplus F_{L_\alpha}$ represents the set of test vectors for fault α. This set is empty if H is 0, when both functions are equal. In such case, the fault cannot be propagated through L. In the other case, all the paths that lead to terminal node 1 (this node represents the logic value 1) are test vectors.

We find a test vector by making the boolean function of the faulty circuit different from the boolean function of the fault-free circuit. To generate a test vector, we traverse the two BDD's corresponding to these boolean functions with the same variables assignment until the terminal nodes reached by both BDD's are different. While traversing the BDD's, we compare the faulty and fault-free BDD's given an assignment. Then, if the logic value 1 is assigned to the i^{th} primary input, we follow always the edge 1 (high) of

any vertex whose index is i while traversing the BDD's. If, for example with $x_i = 1$, the BDDs are different, we continue the traversal. In the opposite case, no test vector can be produced using such assignment. Then, we must change the variable assignment. We set x_i to 0 and repeat the experiment. If with this assignment, the BDDs are also equal, we change the assignment of the variable x_{i-1}. We proceed this way until we reach different terminal nodes. Using this method, a test vector is produced in a linear time.

It is interesting to note that in our approaches we compute the BDD of the faulty circuit and that of the faulty one at the immediate dominator line L of the faulty line. If the two BDD's are different at L, we compute and compare the two BDD's at L. We proceed this way until a primary output is reached. Note that, as soon as the computed BDD's are equal at a dominator line D, the fault can not be propagated through D, we stop the computation.

Example 1: Suppose that we have to test fault G_{s-a-1} (line G stuck at 1) of the circuit shown in Figure 3.2. As indicated in this figure, if we consider only the circuit of S1, H is the immediate dominator of G. S1 is also a dominator of G. if we consider only the circuit of S2, S2 is the only dominator of G.

To find a test vector for the fault G_{s-a-1}, first, at the dominator line H, we compute the boolean functions corresponding respectively to the fault-free circuit and the faulty one with the fault G_{s-a-1}. Figure 3.3 represents the fault-free BDD and the faulty BDD due to the fault G_{s-a-1} at line H. As shown in Figure 3.3, the corresponding BDD's of both functions will be equal, since the graphs represented in the figure are isomorphic. As a result, the functions are equivalent, which means that the fault G_{s-a-1} cannot be propagated through H to the primary output S1.

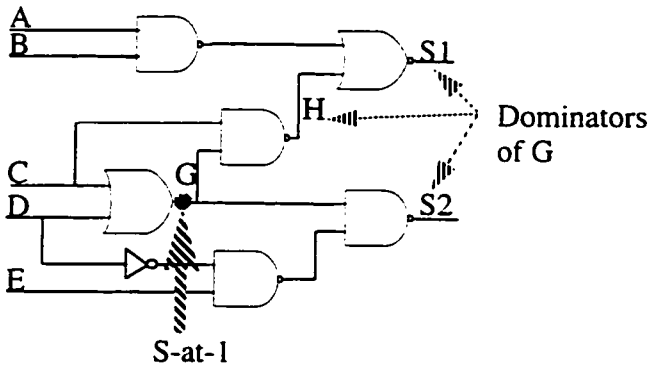


Figure 3.2 A two-output circuit with fault G_{s-a-1}.

Note, however, that we cannot conclude that G is redundant yet, but we must first compute the BDD of the faulty circuit and that of the fault-free one at line S2 (in this case, the second PO of the circuit).

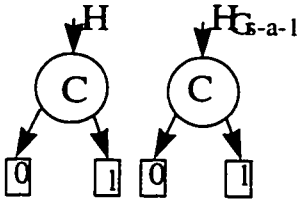


Figure 3.3 BDD of the fault-free circuit and that of the faulty circuit at line

The faulty BDD and the fault-free BDD at line S2 are shown in Figure 3.4. To generate a test vector, the procedure get_a_test() will be called. First, this procedure assign the value 1 to D. According to Figure 3.4, S2 BDD and $S2_{G_{s-a-1}}$ BDD will lead to the same vertex (the terminal node 0). As a result, D must take the value 0. After that, the procedure assigns the value 1 to C. Note that by assigning this value to C, both BDD's will leads to the same vertex E. Then, C must take also the value 0. In this case S2 BDD will lead to the

terminal node 1. Then $S2_{G_{s-a-1}}$ BDD must lead to terminal node 0. Consequently, E must take the value 0.

According to Figure 3.4, we note that we reach different terminal nodes only when C = 0, D = 0 and E = 0. In the BDD of S2, the path (D=0, C=0) will lead to terminal node 1 and, in the BDD of the faulty circuit, the path (D=0, E =0) will lead to terminal node 0. As a result, ABCDE = XX100 (X means "don't care") is a test vector for fault G_{s-a-1}.

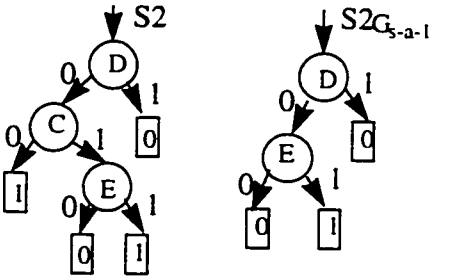


Figure 3.4 BDD of the fault-free circuit and the BDD of the faulty circuit

To generate test vectors, we call the routine Tes_Gen_SO (test vector generation with single ordering for all POs), shown in Figure 3.5. This routine selects an undetected fault and tries to propagate it to a primary output. It calls the routine Select_a_vector() with two arguments, the fault F and the primary output P to which the fault is to be propagated. The procedure Select_a_vector() computes the BDD of the faulty circuit and that of the fault-free one at a line L, the immediate dominator of the faulty line. If this line is not a primary output and the BDD's corresponding to these functions are equal at L, there is no need to compute the boolean function of the fault-free circuit and that of the faulty one at any dominator line of L, since these two functions will be equal at these lines. Note that as soon as the BDD's become equal Select_a_vector() stops the computation and return the

NULL pointer to indicate that the fault can not be propagated to P. This way, we avoid unnecessary computation. Now, if the computed BDD's are different at all the dominators, a test vector is generated in a linear time by calling the procedure Assign_inputs() with arguments the two BDD's computed at the primary output P. If all the outputs are studied and the fault remains undetected, then it is a redundant fault.The three routines are presented in Figures 3.5 and 3.6.

```
Test_Gen_SO ()
 {
  - choose an ordering to the input variables
  For each undetected fault F {
    For(i =0; i < number_of_outputs; i++) {
     Test_vector = Select_a_vector(F,Po[i]);
      if Test_vector is  not NULL {
        - Remove F from the list of undetected faults
         - Fault simulation
          break;
      }
     If(i == number_of_outputs)
        declare F redundant;
    }
  }
}
```

Figure 3.5 The Procedure Test_Gen_SO

Note that when the fault is stuck at 1, to generate the BDD of the faulty circuit at a primary output, the procedure const_1_bdd() is used. In the other case, the procedure const_0_bdd() is used. The procedure const_0_bdd(), as well as const_1_bdd(), is a recursive procedure which uses a depth first traversal of the circuit from the primary output to the primary input to generate the BDD of the faulty circuit at this output. During the traversal, if the line reached is a primary input, its BDD is returned. If the faulty line is reached the terminal node 1 is returned. At the other lines, we compute the BDD.

```
Select_a_vector(F,Po)
Input: A fault F and a line L at which the faulty BDD and the fault-free BDD
       will be computed
Output: A test vector represented by a chained list
{
 i = dominator(F->line,Po);
 if(F is stuck at 0)
  {
   for(; ;){
     faulty_bdd = const_0_bdd(i);
     fault_free_bdd = const_bdd(i);
     if(faulty_bdd == fault_free_bdd)
       return(NULL);
     if(line i is a primary output){
      test_vector =  Assign_inputs(faulty_bdd, fault_free_bdd);
      return(test_vector);
     }
    i = dominator(i,Po);
   }
  }
  else {
   for(; ;){
     faulty_bdd = const_1_bdd(i);
     fault_free_bdd = const_bdd(i);
     if(faulty_bdd == fault_free_bdd)
       return(NULL);
     if(line i is a primary output){
      test_vector = Assign_inputs(faulty_bdd, fault_free_bdd);
      return(test_vector);
     }
     i = dominstor(i,Po);
    }
  }
}
```

Figure 3.6 Procedure select_a_vector()

3.4.2 Test vector generation using variable ordering for each PO

The disadvantage of the first approach is that it may not be possible to find a single good ordering for all the primary outputs in the large examples (c2670 and c7552). To study large circuits we changed our test strategy. Instead of choosing an undetected fault and trying to propagate it to a primary output, we studied the primary outputs separately.

```
Test_Gen_VO()
 {
   For(i = 0; i < number_of_PO; i++){
    order_inputs(i);
    compute_observability(i);
    For each undetected Fault F {
     If (observability[F->line][i] == 1) {
      Test_vector = Select_a_vector(F,i);
      if(Test_vector is not NULL) {
       - Remove F from the undetected faults list
       - Fault simulation
      }
     }
    }
  }
 - Declare all the undetected faults redundant;
 }
```

Figure 3.7 The procedure Test_Gen_Vo

Figure 3.7 shows the procedure Gen_test_VO(). This procedure is used by BDD_FT-EST to generate test vectors by studying the primary outputs separately. As shown in this figure, after selecting a primary output, we order the input variables of the circuit and compute the observability of all the lines of the circuit as described in Section 3.3. We take a fault F and check the observability of its faulty line. If the observability of this line is equal to 1, we try to propagate the fault to this PO. First, we compute the BDD corresponding to the faulty circuit and that corresponding to the fault-free circuit at the immediate dominator D of the faulty line. If the generated BDD's are equal, we stop the computation, the fault can not be propagated to the primary output in question. In the opposite case, we compare the BDD's at the immediate dominator of D. We proceed this way until we reach the primary output or we find that the fault can not be propagated to a dominator line of the faulty line. If the BDD of the fault-free circuit and that of the faulty circuit are different at the primary output, we find an assignment to the primary inputs so that these BDD's will have different terminal nodes. We call the procedure *Select_a_vector()* of Figure 3.6, the

same procedure used by our first approach, to generate a test vector. This routine takes as arguments the BDD of the fault-free circuit and that of the faulty one. Recall that to obtain a test vector, we traverse both BDDs until the terminal nodes reached by both BDD's will be different.

According to Figure 3.7, the test vector generation ends when all POs have been studied. The undetected faults are declared redundant, since they cannot be propagated to any primary output. For maximum efficiency, only the BDD of the primary output and those of the fanout nodes are kept. All the other BDD's are freed.

3.4.3 Acceleration of test vector generation

To accelerate test vector generation using the second approach, we have made two refinements to our algorithm.

Suppose that we have to generate a test vector to the fault α (D stuck at 1) of circuit C shown in Figure 3.8. Note that the presence of α does not affect any line of subcircuit C1, since there is no path between line D and any line of C1. As a result, the BDD of the fault-free circuit and that of the circuit with fault α will be equal at any line of C1. Consequently, while constructing the BDD's for the faulty circuit, we need to generate the BDD's only for circuit lines that can be affected by the fault α. For the other lines, the BDD of the faulty circuit is equal to that of the fault-free one.

To obtain the lines that can be affected by the fault, we traverse the circuit from the faulty line to the primary outputs. All the lines reached will be considered as affected by the fault. Using this refinement, we have reduced the CPU time by almost a factor of 2.

The second refinement can be summarized as follows: Suppose that we have to find a test vector for the fault α and A is the immediate dominator of its faulty line. Suppose also that when we have studied the primary output S1, we have found that the BDD of the faulty circuit and that of the fault-free one are equal at line A. As a result, the fault cannot be propagated to primary output S1. Since we compare the BDD of the faulty circuit and that of the fault free one only at the dominators, while studying the primary output S2, to generate a test vector, we have to construct the BDD of the faulty circuit and that of fault-free one at line A with another variable ordering. In addition, whatever the ordering of the input variables is, these two BDD's will be equal at line A. To avoid this unnecessary computation, in our second approach, for each fault remained after random test vector generation using *fsim* [14], we have a chained list that contains the dominators at which the boolean difference is 0. So, given a fault F and a primary output P, before constructing the BDD's at a dominator D, we check if D exists in the chained list corresponding to F. If it is the case, this fault cannot be propagated to the primary output in question, we conclude that F can not be propagated to P and we try to propagate to P another undetected fault. Using this method, we have reduced the CPU time by a factor of 10 for the circuit c7552 and by almost a factor of 4 for the other circuits.

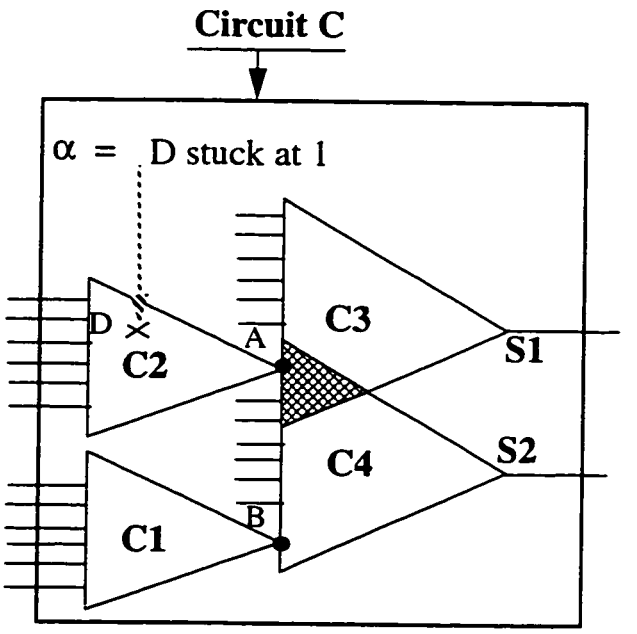


Figure 3.8 A multiple-output circuit C

3.5 Experimental Results

BDD_FTEST is implemented in the C programming language. The results were obtained by running BDD_FTEST on a SUN/4 workstation.

After fault-collapsing, two phases of test pattern generation follow: random and algorithmic. The first phase of test pattern generation is the random phase: We use the fault

simulator *fsim* [14]. In this way, we generate patterns for the easily tested faults (generally 80% to 99% of the total faults).

Table 3.1 Result obtained by running the fault simulator *fsim*

Circuit	Total Faults	Collapsed Faults	Detected Faults	Number of Vectors	CPU(s)
c432	864	524	502	57	0.333
c499	998	758	713	54	0.333
c880	1660	942	878	72	0.550
c1355	2710	1574	1405	60	0.783
c1908	3816	1879	1500	45	1.25
c2670	5340	2239	2239	89	1.3
c3540	7080	3428	2850	97	2.76
c5315	10630	5350	5085	106	3.07
c7552	15104	7550	6785	131	5

Table 3.1 shows the results obtained by running the parallel random fault simulator *fsim*. It shows, for each circuit, the number of faults (Total Faults), the number of collapsed faults, the number of detected faults, the number of test vectors and the CPU time. When *fsim* completes the second phase, algorithmic test pattern generation begins by calling the BDD_FTEST.

During the algorithmic pattern generation phase, each pattern generated is simulated (using a single fault propagation simulator) so that any faults detected by the new pattern may be removed from the fault list.

Table 3.2 Test Generation with the same ordering for all primary outputs

CIRCUIT	BDD_FTEST					TSUNAMI [13]	
	Remaining Faults	Faults P.R.	Nbr of Vectors	Total nbr of Vectors A.C	CPU(s)	Nbr Vect	CPU(s)
c432	22	4	11	50	5.78	39	14.5
c499	45	8	22	56	120	78	178
c880	64	0	34	75	8.467	25	48
c1355	169	8	52	89	160	111	560.6
c1908	379	9	140	133	280.53	121	1307.8
c2670	-	-	-	-	-	-	-
c3540	578	137	140	179	6400	-	-
c5315	265	59	86	140	161.17	71	177.5
c7552	-	-	-	-	-	-	-

Table 3.2 compares our results and those of TSUNAMI, in the case of single ordering for all the outputs. This table shows, for each individual circuit, the number of faults remaining after random generation, the faults proved redundant (Faults P.R), the number of test vectors (Nbr of Vectors) generated for the remaining faults, the total number of vectors after compaction (Total Nbr of Vectors A.C, this number includes the number of vectors generated by *fsim*).

The other two columns present the results reported in [13]. As indicated in this table, the time spent by BDD_FTEST is always much less than that spent by TSUNAMI. It is also interesting to note that BDD_FTEST is able to generate test vectors for the circuit c3540, but TSUNAMI is not. Note that test vectors cannot be generated for the circuits

c2670 and c7552 using a single ordering to all PO. For the other benchmark circuits, we have succeeded in generating test vectors in a short time.

Table 3.3 Test Generation using an independent ordering for each primary output

Circuit	Remaining Faults	Redundant Faults	Nbr of Vectors	Total Nbr of Vectors A.C	CPU(s)
c432	22	4	13	52	10.01
c499	45	8	22	56	304
c880	64	0	32	71	6.2
c1355	169	8	52	89	278
c1908	379	9	159	138	83
c2670	508	117	96	117	532
c3540	578	137	145	190	10000
c5315	265	59	86	147	55.13
c7552	765	131	155	240	95

Table 3.3 shows the results when the input variables take different ordering for each primary output. It is interesting to note that all the ISCAS'85 benchmark circuits have been studied. Example c6288 is not studied since it is easy to test using the random fault simulator *fsim*. It is a 32-bit multiplier for which it has been proven that the ROBDD must always be of exponential size for some outputs, even for optimum input ordering.

Table 3.4 Test Generation for some sequential circuits

Circuit	Collapsed Faults	Nbr Faults P.R	Nbr of Vectors A.C	CPU(s)
s298	308	0	40	0.3
s420	455	0	75	1.6
s444	474	14	38	0.5
s713	581	38	38	5.07
s832	870	14	120	2.37
s953	1079	0	100	3
s1488	1486	0	131	3.67

Table 3.4 Test Generation for some sequential circuits

Circuit	Collapsed Faults	Nbr Faults P.R	Nbr of Vectors A.C	CPU(s)
s1494	1506	12	13	4.02
s5378	4503	40	280	4.02
s15850	11725	389	560	3000
s35932	39094	3984	75	930

It is interesting to note from Table 3.2 and 3.3 that in the examples for which we could find a single ordering, the CPU time was in general less than that in the case with a separate ordering for each PO. We generate test vectors for the same set of faults that remain undetected after running *fsim*, we use the same seed and the same number of vectors generated randomly.

Note that, by using the second approach we have succeeded in generating test vectors for all the benchmark circuits, which is not the case using TSUNAMI or our first approach.

Table 3.4 gives results for some ISCAS89 benchmark circuits [12]. ISCAS'89 were processed under the full scan assumption.

3.6 Conclusion

We have presented an algebraic technique for generating tests for single stuck-at faults in combinational circuits. Experimental results with ISCAS'85 benchmarks show that this approach is well-suited to test generation for very hard or redundant faults in a circuit.

This method has many advantages over previously proposed algebraic methods. It uses the dominator notion to eliminate backtracks and to accelerate test vector generation. Moreover, given the BDD of the faulty circuit and that of the fault-free one, a test vector

can be found in a linear time. Then, the existence of a test can be determined easily. In addition, by using variable ordering for each primary output, we can study many benchmark circuits that cannot be studied using a single ordering for all primary outputs.

CHAPITRE 4

Génération hiérarchique de vecteurs de test basée sur le BDD et sur un concept nouveau d' observabilité

Introduction

Le présent chapitre est basé sur un papier de conférence intitulé "*Hierarchical Algebraic ATPG Based on BDD Representation and on a New Observability Concept*" qui a été soumis à *IEEE Transactions on Circuits And Systems*.

Dans le quatrième chapitre, nous présenterons une autre nouvelle approche pour générer des vecteurs de test pour les circuits combinatoires. Elle est basée aussi sur les diagrammes de décisions binaires (BDD). Celle-ci permet d'étudier toutes les pannes non-détectées correspondant à la même porte logique, en ne déployant presque aucun effort additionnel. Dans ce chapitre, les sorties primaires seront étudiées séparément pour générer des vecteurs de test pour plus de circuits.

Pour simplifier, considérons un circuit avec une seule sortie primaire. La stratégie de test diffère de celle décrite précédemment dans le chapitre 3. C'est une méthode hiérarchique qui est basée sur l'observabilité d'une ligne par rapport à la sortie primaire. Ici, l'observabilité (O_l^P) n'est pas une valeur réelle qui indique la probabilité d'observer la ligne l

à la sortie P, mais plutôt une fonction booléenne qui regroupe toutes les assignations possibles qui font de sorte que la valeur à la sortie primaire ne dépend que de la valeur à la ligne considérée. Notons qu'une erreur résultante d'une panne à la ligne *l* se propage à une sortie primaire P si et seulement si $O_l^P = 1$.

Rappelons que les deux conditions à satisfaire pour arriver à tester une panne sont: l'activation de la panne et la propagation de l'erreur résultante à une sortie primaire. Notons que $f_l \cdot O_l^P$, où f_l est la fonction logique à la ligne *l* du circuit et O_l^P l'observabilité de la ligne *l* par rapport à la sortie P, représente l'ensemble de vecteurs qui testent le défaut logique *l c-à-0* (collée-à-0). Il est à noter que $\bar{f}_l \cdot O_l^P$ est égale à 1 si et seulement si f_l est égale à 0 et O_l^P est égale à 1. Lorsque f_l est égale à 1, la panne collée à 0 sera activée, et le fait d'avoir O_l^P égale à 1 implique que l'erreur résultante sera propagée à la sortie P. De même, $\bar{f}_l \cdot O_l^P$ représente l'ensemble de vecteurs qui testent la panne *l collée-à-1*.

En utilisant notre technique de manipulation des fonctions, présentée au chapitre 2, le complément d'une fonction booléenne quelconque sera trouvé dans un temps négligeable. Dans ce travail, nous montrerons que, si f_l et O_l^P sont disponibles, l'ensemble de vecteurs de test pour la panne *l collée-à-0* et celui pour la panne *l collée-à-1* seront trouvés dans un temps approximativement égal à celui requis pour une seule panne.

Quant au test hiérarchique, on utilise la notion de "*supergates*" [26] pour propager une erreur vers une sortie primaire ou bien pour trouver une assignation aux entrées primaires permettant d'avoir la valeur logique qu'on désire sur une entrée ou une sortie d'un *supergate*. Le circuit composé de "*supergates*" sera sans sortance. Par conséquent, on peut donc faire des assignations aux entrées des *supergates* sans générer aucun conflit entre les signaux. En d'autres termes, sans assigner à la même ligne deux valeurs logiques dif-

férentes en même temps. Signalons qu'en utilisant la notion de *supergates*, les tailles des BDD à manipuler seront réduites. Ceci provient du fait que les fonctions correspondantes aux BDD générés ne seront exprimées qu'en fonction des entrées des *supergates*. Notons que, dans une génération de BDD normale, chaque noeud du BDD correspond à une entrée primaire du circuit et, généralement, les BDD correspondant aux entrées des *supergates* sont de taille supérieure à 1. Par conséquent, les BDD, qui ne contiennent que des entrées primaires seront de plus grandes tailles. En d'autre termes, à la place du noeud correspondant à une entrée d'une *supergate*, on trouvera le BDD de cette entrée qui ne contient que des noeuds liés aux entrées primaires du circuit.

Hierarchical Algebraic ATPG Based on BDD Representation and on a New Observability Concept

Bechir AYARI and Bozena KAMINSKA
Ecole Polytechnique de Montréal
Department of Electrical and Computer Engineering
P.O Box 6079, Station Centre-ville
Montréal, PQ, H3C 3A7
Email: bozena@vlsi.polymtl.ca
Tel: (514) 340-4270

Abstract :

This paper presents a hierarchical test generation technique based on OBDD representation and on a new observability concept. The observability of a gate, a boolean function represented by an OBDD, gives all the PI assignments that makes the value at the output of this gate observable at a primary output. Using this concept, all the undetected faults related to a gate can be studied with almost no more effort than studying only one fault. The concept of dominance is used for identifying the set of maximal supergates. This enables hierarchical generation of test vectors for stuck-at faults and accelerates test vector generation, since smaller BDDs will be manipulated. In our approach, the PO are studied separately to generate test vectors for larger set of networks than existing methods

based on BDDs. For all the circuits analyzed, the algorithm is faster than previous algebraic methods. It is suitable for complex circuits, like c7552 and c2670, which cannot be studied using the BDD approaches presented in [13] and [25]. It is also very efficient for hard-to-detect and redundant faults.

4.1 Introduction

Test pattern generation can be viewed as a branch-and-bound problem [17]: test pattern generation algorithms usually search through the space by systematically branching and bounding until a test pattern is discovered or the search space is exhausted.

To reduce the solution space, PODEM [17] and FAN [18] algorithms identify some necessary assignments using local implication. The contribution of SOCRATES [19] was to find additional necessary assignments that could not be found using local implication. Rajski [16] has presented a test pattern generation algorithm that uses the concept of necessary assignment. A 16-value algebra is employed to reduce or eliminate backtracking in automatic test vector generation. However, there typically exist some so-called "hard" faults that resist such techniques and lead to unacceptably long test generation times when very high single stuck-at fault coverage is desired.

Other techniques use algebraic methods, principally boolean difference, to determine the entire test set for a fault [20], [21] and [22]. Early algebraic methods were very slow compared to the enumeration techniques. Recently, however, algebraic methods have been proposed that are fast enough to be useful in generating tests for hard faults that resist these techniques.

In the Larrabee method [21], the product of sums representation of the boolean difference is first found in terms of the primary inputs and outputs of the gate in the circuit. This is followed by a search to satisfy assignment of this formula. In [23], Smirat *et al.* have presented a transitive-closure-based test generation algorithm. A test is obtained by determining signal values that satisfy a Boolean equation derived from the neural network model of the circuit incorporating the necessary conditions for fault activation and path sensitization.

TSUNAMI algorithm [13], on the other hand, is an algebraic method using BDDs. This algorithm involves a path-oriented search similar to the one used by conventional enumeration algorithms. The method determines the set of input assignments that will propagate a fault through a gate in a path from the fault site to a primary output. This makes it possible to stop the computation as soon as a fault is either propagated to some primary output or it becomes unobservable. The set of tests is defined in terms of primary inputs only.

Other techniques for gate-level ATPG using OBDDs have been presented in [25]. Such systems use the path-sensitization-based TPG to generate tests for the easy faults and the OBDD-based generator for hard and redundant faults.

The disadvantage of the techniques presented in [13] and [25] is that they may not make it possible to find a single good ordering for all the primary outputs. This is the reason why benchmark circuits c2670 and c7552 [7] cannot be studied using these methods. In addition, to generate test vectors using these techniques, the difference in representation of good and faulty circuits must be found.

In this paper we present a hierarchical test vector generation method based on *observability*, a new concept whose definition is given in Section 4.4. The approach involves studying the primary outputs separately. A variable ordering is chosen for each PO, depending on its circuit topology. In this way, more complex circuits can be studied and smaller BDDs will be manipulated.

Given a fault, to generate a test vector, we need to compute the difference in representation of good and faulty circuits only at the supergate output corresponding to the faulty line. Next, the fault is propagated by making the supergate output observable at the primary output. Observability is a boolean function that holds all the PI assignments that allow us to make a line Li observable at a line Lk. Using the observability concept, once a fault at line L (input or output) of a gate G is studied, the other faults related to the same gate can be studied with almost no additional effort.

Each line of the circuit is expressed in terms of its supergate inputs, since they are mutually independent. Then, the BDDs generated will contain only nodes corresponding to the supergate inputs. In this way, the BDDs to be manipulated will be reduced and the time spent to construct BDDs and to generate test vectors will also be reduced.

The paper is organized as follows: In Section 4.2, the basic concepts related to BDDs are reviewed. In Section 4.3, a decomposition method [26] based on the dominance concept is presented. Section 4.4 introduces the observability concept. In Section 4.5 our hierarchical test vector generator is discussed. We give some experimental results in Section 4.6 and conclude in Section 4.7.

4.2 A Review of BDDs

Some BDD-related terms are reviewed here. These terms have been defined in [1] and are explained now for easy reference in subsequent sections. Consider the decision graph in Figure 4.1 for the boolean function F = X1.X2 + X3. Each vertex in the graph corresponds to an input variable. To evaluate F at a given input vector $i = (i_1, i_2, i_3)$, we traverse the graph from the root to the leaves in the following manner: At a vertex with index j, we take the 0 (low) branch if i_j is 0 and the 1 (high) branch if i_j is 1. This process continues until we reach a terminal vertex (0 or 1),which is the value of f for that input vector.

Two BDDs with the same variable ordering represent the same function if and only if the two graphs are isomorphic. This test is very simple to do. Using our implementation for generating ROBDDs, we guarantee that the BDDs of equivalent functions will be indicated by the same pointer. This is called a strong canonical form. The test for equivalence is then a constant-time operation. In [1], Bryant has also presented a polynomial time algorithm (in the size of the BDDs) for manipulating boolean functions. In particular, if we have BDDs for the functions F and G and a binary boolean operator $\Diamond$. We may then obtain the BDD for the function $F \Diamond G$ in polynomial time.

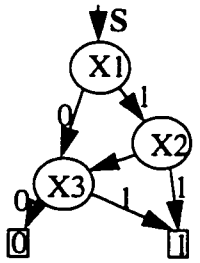


Figure 4.1 An ordered binary decision diagram for the function F = X1X2 +X3

4.3 Circuit Decomposition

The *supergate* was used by Sharad *et al.* [26] to obtain an exact analysis for efficient computation of random-pattern testability in combinational circuits. In our case, determination of the maximal supergates accelerates the BDDs generated, since any line of the circuit will be expressed only in terms of the inputs of the corresponding supergate. In this way, the size of BDDs to manipulate will be smaller, and test vectors can be obtained hierarchically.

4.3.1 Supergate Definition

A formal definition of the supergate is introduced in [26]. The supergate of a line X in the circuit, denoted by G(X), is the smallest predecessor subnetwork feeding X whose inputs are logically independent, i.e. have no signal correlation. The output node of a supergate SG(X) is the node X itself. A supergate example is given in Figure 4.2.

Figure 4.2 shows a NAND circuit with the supergate of the primary output shown in the shaded triangle. In the example, lines *l0, l1* and *l2* define supergate inputs and *l11* the supergate output.

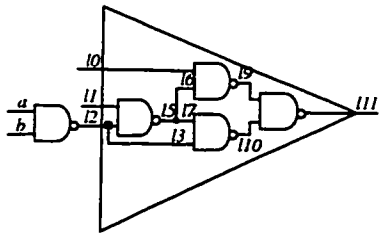


Figure 4.2 Example of a supergate

A supergate is said to be maximal if it is not properly contained in a larger supergate. As an example, for the network in Figure 4.3, the circuit graph and the maximum supergates (shown within dotted lines) are given in Figure 4.4.

4.3.2 Identification of Maximal Supergates

Let R(X) denotes the set of all nodes in G, from which node X is reachable by a directed path. Each supergate represents a minimum subcircuit with logically independent inputs. A more precise statement of the same idea is that for every interior node *i* in a supergate SG(X), there exists another interior node *j* in SG(X), such that R(i) and R(j) both include at least one common (fanout) input of SG(X). The concept of dominance can be used [26] for identifying the set of maximal supergates in a combinational circuit.

Given a directed graph G(V,E) with a distinguished source vertex *s*, we say that a node v_i dominates a node v_j if all directed paths from *s* to v_j also pass through v_i [28]. It is known that the dominance relation induces a partial order and that the set of nodes which dominate a given vertex is linearly ordered. Because of this linear ordering, the immediate dominator of a node n can be uniquely defined as the dominator that is closest to n on any path from the source s to n. This allows depiction of a dominance relationship among vertices using a tree called the dominator tree [28] in which the source vertex is the root and the predecessor of every other node is its immediate dominator.

Algorithm 1 [26]: To find all the maximal supergates in a circuit graph of a single-output combinational network, proceed as follows:

Step 1: Given the circuit graph G, construct a direct flow graph FG as follows: *(i)* delete the primary-output node and its incident edge. *(ii)* reverse the directions on the edges

which are incident to the node corresponding to the output gate and make all other edges bidirectional.

Step 2: Construct the dominator tree of FG.

Step 3: Start from the root node of the tree and collect all the nodes (including the root) that are children of the root. Include all single successors of these children as well, if any. (This set gives the maximal supergate corresponding to the output node.) Remove all the edges covered so far and iterate the process until the tree is empty.

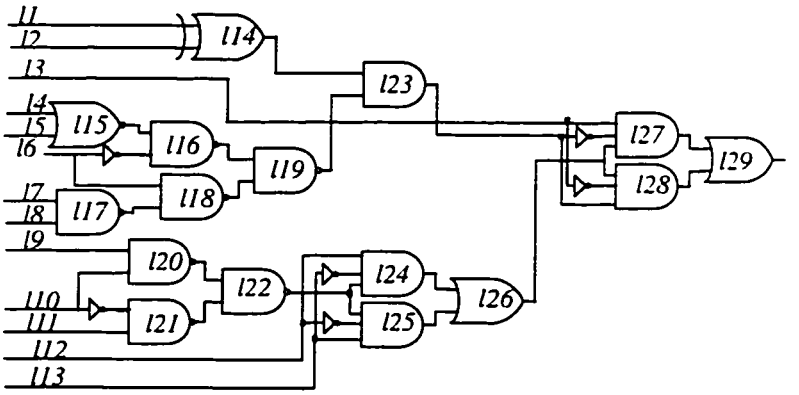


Figure 4.3 Circuit example

Example: Consider the circuit in Figure 4.3. The dominator tree corresponding to its flow graph FG is shown in Figure 4.4. According to this figure, the maximal supergates are: SG(*l29*), SG(*l26*), SG(*l22*), SG(*l23*), SG(*l19*), SG(*l14*), SG(*l15*) and SG(*l17*).

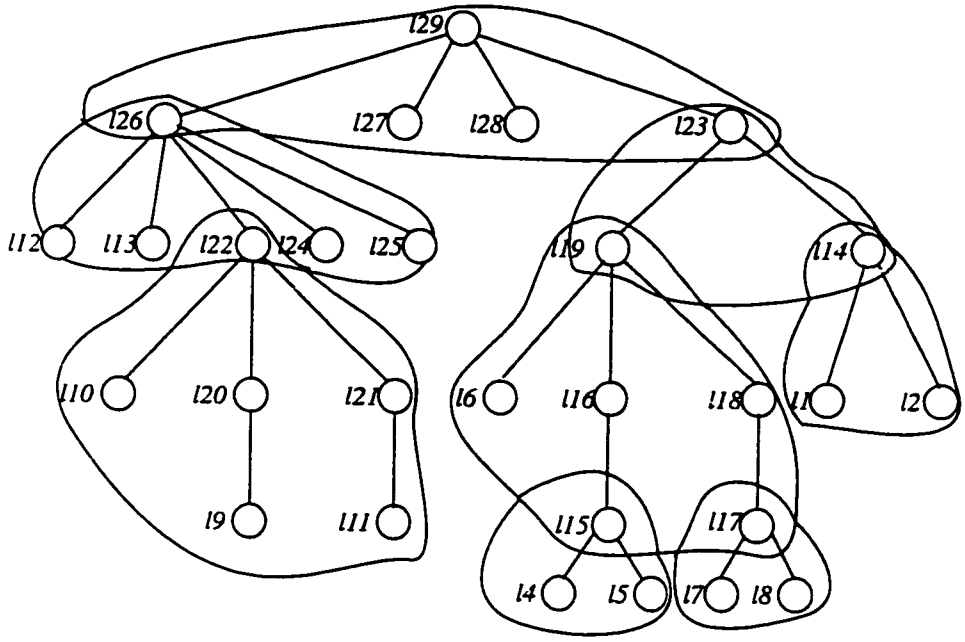


Figure 4.4 Dominator tree and maximal supergate partitioning

For a single-output combinational circuit, there exists an interesting topological relationship among the maximal supergates. We define a reduced circuit graph (RCG) as a directed graph (V1,E1), where V1 denotes the set of supergates and a directed edge (i,j) exists if the output node of SG(i) is an input node of SG(j). Note that the RCG is a tree, i.e. for every pair of nodes in the RCG there is no more than one directed path. The RCG of the circuit in Figure 4.3 is shown in Figure 4.5b.

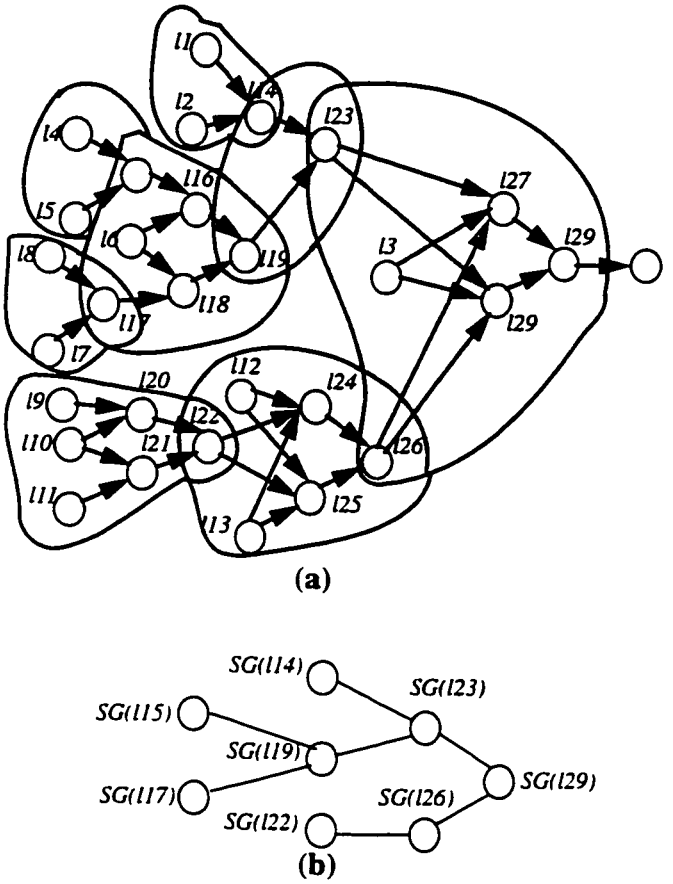


Figure 4.5 (a) Circuit graph and maximal supergates, and (b) Reduced circuit graph(RCG) of the circuit shown in Figure 4.3.

4.4 Observability Definition

Definition 1: The observability of a line *L1* at a line *L2*, denoted by O_{L1}^{L2}, represents all the assignments that make the logic value at line *L2* depends only on the logic value at line L1.

Then, O_{L1}^{L2} represents the set of assignments that makes f_{L2} (the boolean functions at lines L2) equal to either f_{L1} (the boolean functions at lines L1) or $\overline{f_{L1}}$. And, when O_{L1}^{L2} is

equal to 1, the logic value at line *L1* can be deduced by observing the logic value at line *L2* only.

4.4.1 Observability Computation

The OBDD representation is used to compute the observability of any line of the circuit. As will be seen later, the easiest way to compute the observability of a line *L* is to suppose that this line is a primary input that is first in the BDD ordering.

Suppose that we have an extra primary input called D that is first in the OBDD ordering. In other words, the lines of the circuit, including the PIs, are ordered after D. The input D is only introduced to make the observability computation much easier and more efficient. The circuit topology remains unchanged.

To compute the observability of line *Li* at line *Lk*, the OBDD of *Lk* is constructed first on the assumption that *Li* is the extra primary input D. Then, when line *Li* is reached while traversing the circuit, beginning from line *Lk* to the PIs for computing O_{Li}^{Lk}, the BDD of D is returned. The BDD computed then corresponds to the boolean function at line *Lk*, on the assumption that *Li* is a PI called D. This function is given by the following equation:

$$F_{Lk} = \overline{D} \cdot f_0 + D \cdot f_1 \tag{4.1}$$

Note here that when $f_0 = f_1$ the logic value at line *Lk* does not depend on D (Li) and, consequently, the OBDD of F_{Lk} will not contain a node corresponding to the extra input D. In this case, line *Li* cannot be observable at line *Lk*, whatever the primary input assignments are. In other words, a fault at line *Li* cannot be propagated through line *Lk*. In the opposite case, when $f_0 \neq f_1$, node D is the root node of the BDD constructed (Lk OBDD), since D is first in the BDD ordering. It refers to f_0 BDD and f_1 BDD by its low and high

edges respectively. In this case, there exists at least one assignment to the primary inputs that allows propagation of a fault from line *Li* through line *Lk*. The observability of line *Li* at line *Lk* is given by the following equation:

$$O_{Li}^{lk} = f_0 \oplus f_1 \tag{4.2}$$

Note that O_{Li}^{lk} does not depend on the "added" primary input D, and that once O_{Li}^{lk} is known the boolean function F_{Lk} will never be needed and the corresponding BDD can be freed for maximum efficiency. Note, too, that according to Equation 4.2, when $O_{Li}^{lk} = 1$, $f_0 \oplus f_1 = 1$. Consequently, according to Equation 4.1, F_{Lk} is equal to D or to $\overline{D}$. We then have $F_{Lk} = f_{Li}$ or $F_{Lk} = \overline{f_{Li}}$. Consequently, the logic value at line Li can be deduced by observing the logic value at line Lk only.

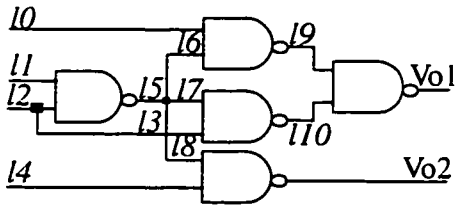


Figure 4.6 A two-output example

Example: Suppose that we have to compute O_{l3}^{Vo1}, the observability of line l3 at the output Vo1 of the circuit shown in Figure 4.6. To obtain O_{l3}^{Vo1}, we first compute the boolean function of line Vo1 by supposing that *l3* is the primary input D (Figure 4.7a). In this case, $Vo1 = \overline{D}(l0 \cdot (\overline{l2} + \overline{l1})) + D(\overline{l2} + \overline{l1})$ and $O_{l3}^{Vo1} = \overline{l0} \cdot \overline{l2} + \overline{l0} \cdot \overline{l1}$.

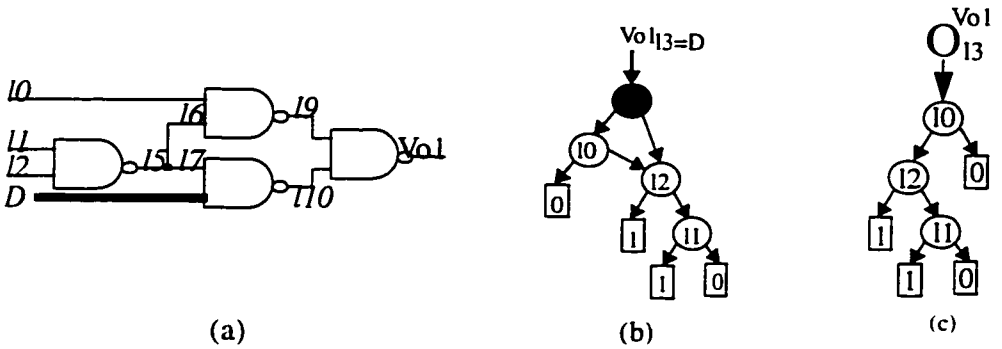


Figure 4.7 (a) circuit Vo1 with l3 = D, (b) the BDD of Vo1, and (c) the observability of l3 at Vo1

Note that O_{l3}^{Vo1} is equal to 1 when *l0*, *l1* and *l2* are equal to 0 X 0 or 0 0 X respectively. Indeed, these two assignments make it possible to observe line *l3* at Vo1 *(Vo1=D)*. In other words, with such assignments the logic value at Vo1 is always equal to the logic value at line *l3*.

4.4.2 Observability of a Gate Input

The observability of a gate input can be obtained in two different ways. The first manner has been described in Section 4.4.1. The second is described in this section, where, we will describe how the observability of an input of a gate G can be deduced from the observability of the gate itself.

Let G be a gate of n inputs I_0 to I_{n-1} and an output O. Let P be the primary output of the circuit and F_i be the boolean function corresponding to I_i. Now, to make I_i observable at P, we must make it first observable at line O. The observability of I_i can be computed as follows:

$$O^P_{I_i} = O^O_{I_i} \cdot O^P_O. \tag{4.3}$$

AND gate

To find $O^O_{I_i}$ (the observability of the ith input of an AND (NAND) gate at its output O), O^O_{Ii}, expressed in terms of the gate inputs only, is given by the following equation:

$$O^O_{Ii} = F_0 \cdot F_1 \cdot \ldots \cdot F_{i-1} \cdot F_{i+1} \cdot \ldots \cdot F_{n-2} \cdot F_{n-1}. \tag{4.4}$$

According to Equation 4.4, to observe a primary input of an AND gate at the output of the gate, all the other inputs of this gate must be set to 1 (noncontrolling value of an AND (NAND) gate).

Example: Suppose that we are interested in finding the observability of line *l6* at Vo1, Figure 4.6, using Equation 4.3.

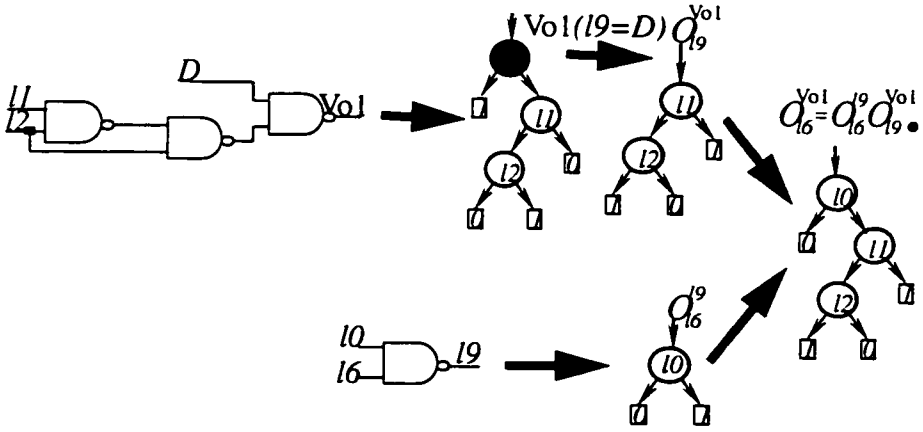


Figure 4.8 Computation of O^{Vo1}_{l6} using Equation 4.3

According to Figure 4.8, $O^{Vo1}_{l9} = l1 + \overline{l2}$. Since *l9* is an output of a NAND gate and *l6* is one of its inputs, then $O^{l9}_{l6} = l0$. $O^{Vo1}_{l6} = l0 \cdot (l1 + \overline{l2})$. Figure 4.9 shows the com-

putation of O_{l6}^{Vol} according to Section 4.4.1. Note that the two ways of computing O_{l9}^{Vol} lead to the same result.

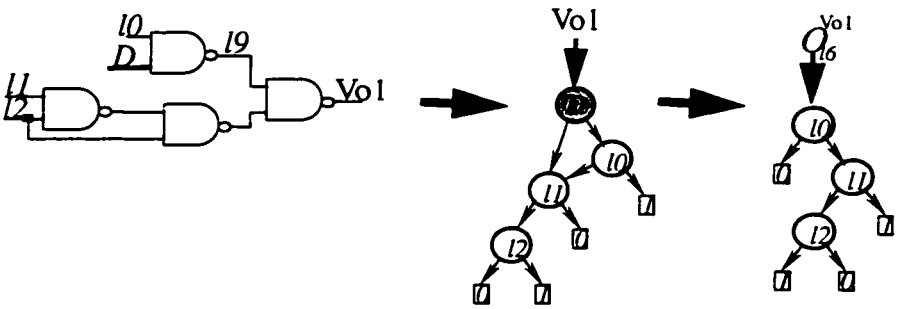


Figure 4.9 Computation of O_{l6}^{Vol} using Equation 4.2

OR (NOR) gate

O_{li}^{O} is given by the following equation:

$$O_{li}^{O} = \overline{F_0} \cdot \overline{F_1} \cdot \ldots \cdot \overline{F_{i-1}} \cdot \overline{F_{i+1}} \cdot \ldots \cdot \overline{F_{n-2}} \cdot \overline{F_{n-1}} \tag{4.5}$$

This formula tells us simply that to propagate a fault at the input line l_i of an OR(NOR), all the other inputs of the gate must be made equal to 0 (0 is a noncontrolling value of an OR (NOR) gate).

XOR (XNOR) gate

By supposing that

I_i = D, the boolean function at the output of an XOR (XNOR) gate is always equal to $FO = \overline{D} \cdot f_0 + D \cdot f_1 \;\; with \;\; f_1 = \overline{f_0}$. Then, $O_{li}^{O} = 1$, and line I_i is observable at the output of the gate whatever the logic values at the others inputs are.

4.4.3 The Use of Dominators in Observability Computation

Using the dominators, we can accelerate observability computations. The observability of a line L0 can be computed as shown in the following theorem.

Theorem 1:

$O^P_{L0} = O^{L1}_{L0} \cdot O^{L2}_{L1} \ldots O^{Li+1}_{Li} \ldots O^{Ln}_{Ln-1} \cdot O^P_{Ln}$, where Li is a line of the circuit, Li+1 is the immediate dominator of Li ($0 \leq i \leq n$), and P (Ln+1) is the output of the circuit.

Proof: This theorem simply tells us that to propagate a fault *L0 s-a-v* to P the fault must be propagated through the dominator lines of *L0* (*Li* with ($1 \leq i \leq n$). The question that arises, if one of the observabilities is equal to 0 ($O^{Li+1}_{Li} = 0$), is the following: Is it possible to propagate the fault to P (the primary output)?

If we suppose that $O^{Li+1}_{Li} = 0$: In this case, a fault at line *Li* cannot be propagated to line L_{i+1}. As a result, this fault cannot be propagated to the primary output, since all the paths of Li must pass through L_{i+1} to reach the PO. Since all the paths of *L0* pass through *Li* (*Li* is a dominator of *L0*) to reach the PO, a fault at line L0 must first be propagated to *Li*. Also, since a fault at line *Li* cannot be propagated to the PO, a fault at line *L0* cannot be propagated to the PO.

Example: Suppose that *L1* is a dominator of lines *l0* and *g0*. Suppose that the O^P_{l0} is computed while generating a test vector to a fault at line *l0* using theorem 1, i.e. $O^P_{L0} = O^{L1}_{l0} \cdot O^{L2}_{L1} \ldots O^{Li+1}_{Li} \ldots O^{Ln}_{Ln-1} \cdot O^P_{Ln}$. Suppose that the BDD corresponding to $O^{L_{i+1}}_{L_i}$ has not been freed. Also, according to theorem 1, $O^P_{g0} = O^{L1}_{g0} \cdot O^{L2}_{L1} \ldots O^{Li+1}_{Li} \ldots O^{Ln}_{Ln-1} \cdot O^P_{Ln}$. Note that to obtain O^P_{g0}, only O^{L1}_{g0} must be computed. In this way, the algorithm is accelerated, since all the other observabilities are known.

4.5 Test Vector Generation

The two fundamental steps in generating a test for a fault *l s-a-v* are as follows: first *activate* (excite*) the fault*, and second, *propagate the resulting error* to a primary output (PO). Activating the fault means setting the primary input (PI) values that cause line *l* to have value $\bar{v}$. Propagating the resulting error means making line l observable at one of the primary outputs.

Theorem 2: Any assignment to the PIs that makes $S = (f_{l0} \oplus v) \cdot O_{l0}^{P} = 1$ is a test vector for the fault *l0 s-a-v*, where f_{l0} is the boolean function at line l0 in the fault-free circuit, P is the primary output to which the fault is to be propagated.

Proof: If the assignments chosen for the primary inputs makes $(f_{l0} \oplus v) \cdot O_{l0}^{P} = 1$, then we necessarily have $f_{l0} \oplus v = 1$ and $O_{l0}^{P} = 1$. $f_{l0} \oplus v = 1$ means that f_{l0} is made equal to $\bar{v}$. Fault *l0 s-a-v* is then activated, since $\bar{v}$ and $\bar{v}$ are the values at line l0 in the fault-free and the faulty circuits. Fault-free and faulty circuits will have different logic values at line *l0*. The composite value is then either D or $\overline{D}$ at *l0*. $O_{l0}^{P} = 1$ means, by definition, that P is made equal to either *l0* or $\overline{l0}$. Then, in both cases, the fault is propagated to the primary output P, and we have either D or $\overline{D}$ at P. As a result, the primary input assignments satisfy both conditions of test vector generation, namely fault activation and error propagation, so any assignment that makes *S* equal to 1 is a test vector for the fault *l0 s-a-v*.

According to Theorem 2, therefore, for the fault *l0 s-a-0*, any assignment to the PIs that makes $S_0 = f_{l0} \cdot O_{l0}^{P} = 1$ is a test vector, since $f_{l0} \oplus 0 = f_{l0}$. And, for the fault *l0 s-a-1*, any assignment to the PIs that makes $S_1 = \overline{f_{l0}} \cdot O_{l0}^{P} = 1$ is a test vector, since $f_{l0} \oplus 1 = \overline{f_{l0}}$.

We may recall that O_{l0}^{P} is independent of the kind of fault at line l0, since it is independent of the logic value at *l0* (Equation 4.2). Note that by using complement edges in the BDDs, in addition to making function manipulations easier, the complement of a function f_{l0} is obtained in a negligible amount of CPU time. To generate test vectors for the faults *l0 s-a-0* and *l0 s-a-1*, $S_0 = \overline{f_{l0}} \cdot O_{l0}^{P}$ and $S_0 = f_{l0} \cdot O_{l0}^{P}$ are made equal to 1. Note that in both cases we need to compute O_{l0}^{P} and f_{l0}. If a test is generated to *l0 s-a-v*, then a test vector for fault *l0 s-a-*$\bar{v}$ can be generated in a negligible amount of CPU time.

To test a gate input, the observability of this input at the primary output is computed according to Equation 4.3. Note that to test any fault related to a gate (an input or the output of the gate *s-a-0* or *s-a-1*) we need to compute the observability of the output of the gate at the PO and the boolean functions of all the inputs of the gate, and since the observability of a gate input at the output depends only on the boolean functions of the other inputs and on the observability of the gate itself (Equation 4.3), any other fault related to the gate can be tested with a negligible amount of effort.

Example: Suppose that we are interested in finding a test vector for *l6* s-a-1. Then, we have to make $\overline{f_{l6}} \cdot O_{l6}^{l9} \cdot O_{l9}^{Vo1} = 1$. The observability of line *l6* at Vo1 has been computed in Section 4.4.2, (see Figure 4.8) $O_{l6}^{Vo1} = l0 \cdot (l1 + \bar{l2})$. According to Figure 4.9, $\overline{f_{l6}} = l1 \cdot l2$. *S* is equal to $\overline{f_{l6}} \cdot O_{l6}^{l9} \cdot O_{l9}^{Vo1} = l0 \cdot l1 \cdot l2$. It is equal to 1 only if *l0*, *l1* and *l2* are equal to 1, 1, and 1 respectively. Such an assignment is a test vector for *l0 s-a-1*.

4.5.1 The Use of Supergates in Test Vector Generation

Figure 4.11 shows the structure of our algorithm for generating a test vector for *l s-a-v*. It initializes all values to X and performs the two basic steps of test vector generation

(fault activation and error propagation). Activating a fault means setting primary inputs that choose an assignment resulting in a desired value setting on a specified line of the circuit.

Satisfy() (Figure 4.12) is a recursive procedure in which the value of a supergate output is set to a desired value by choosing an appropriate assignment to the inputs of supergate G1 (a path from the BDD leading to the desired terminal node). If an input of G1 is the output of another supergate G2, *Satisfy()* is called to set the output of G2 to the value needed at the input of G1, and so on, until PIs are reached.

The algorithm is divided into two main steps:

Step 1: Suppose that we have to test a fault l *s-a-v*. Let l belong to supergate SG0 whose output is L0. We need to activate the fault and propagate it first to L0. We activate the fault and propagate the error to the SG0 output to satisfy $S = (f_l \oplus v) \cdot O_l^{L0} = 1$. If S is found to be equal to 0, the fault cannot be activated and propagated to the SG0 output at the same time. In this case, there is no need to go further. In the opposite case, we try to propagate the fault to the PO as described in Step 2.

Step 2: Propagate the error to the considered PO of the circuit: Since the RCG (reduced circuit graph) corresponding to the flow graph of the circuit is a fanout-free circuit, then from any supergate there is only one path leading to the PO considered. There is, therefore, a unique path from SG0 to the PO. Let this path be (SG0, SG1,...,SGi,..., SGn), where SGi is a supergate and P the PO supergate. Let the output of SGi be connected to an input of SGi+1 and SGn be the P supergate. This path will be sensitized (the error is propagated) by satisfying $O_{Li}^{Li+1} = 1$ (making Li observable at L_{i+1}), for $i=0$ to

i=n-1. Note that the algorithm stops computing when $O_{Li}^{Li+1} = 0$, since in this case the fault cannot be propagated to the PO considered.

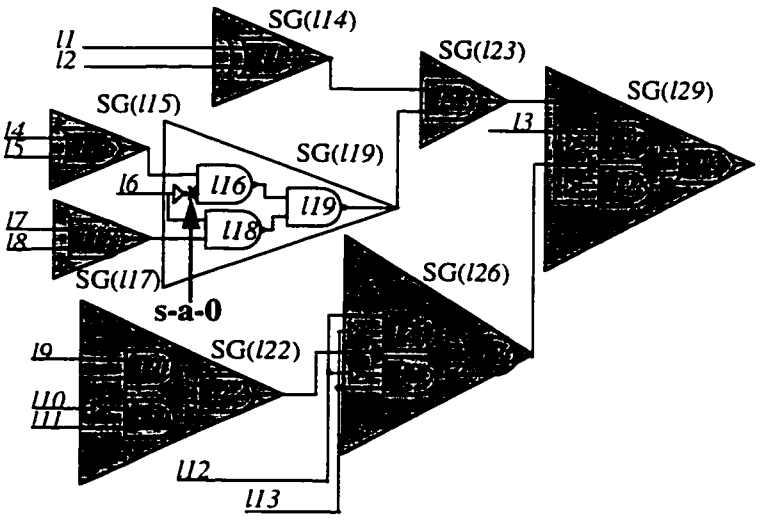


Figure 4.10 The circuit in Figure 4.3 with the second input of gate *l16 s-a-0*

Example: Let us generate a test vector for α (the second input of gate *l16*) s-a-0 (Figure 4.10). In this figure, line α belongs to supergate SG(119). Let the boolean function at a line X be denoted by $F(X)$. The boolean functions corresponding to the supergates are

$F(l19) = \overline{l4} \cdot \overline{l5} \cdot \overline{l6} + l6 \cdot (\overline{l7} + \overline{l8})$, $F(l22) = l10 \cdot l9 + \overline{l10} \cdot l11$,

$F(l23) = l14 \cdot l19$, $F(l26) = l22 \cdot l12 \cdot \overline{l13} + l22 \cdot \overline{l12} \cdot l13$ and

$F(l29) = l26 \cdot \overline{l23} \cdot l3 + l26 \cdot l23 \cdot \overline{l3}$.

According to these equations, each output of a supergate is expressed as a function of its inputs. In this case, the BDD of any line of a Supergate contains only nodes corresponding to that supergate. Consequently, any path chosen from this BDD does not create a conflict, since the supergate inputs are independent. According to theorem 2, a test vector for α s-a-0 must satisfy

$S = f_\alpha \cdot O_\alpha^{l16} \cdot O_{l16}^{l19} \cdot O_{19}^{l23} \cdot O_{l23}^{l29} = 1$. $O_\alpha^{l16} = l15$ $O_{l16}^{l19} = \bar{l6} + \overline{l17}$, $O_{19}^{23} = l14$ and $O_{l23}^{l29} = l26$.

Note that f_α and $O_\alpha^{l19} = O_\alpha^{l16} \cdot O_{l16}^{l19}$ depend on the same inputs (they belong to the same supergate). Since 19 is the output of a supergate (Figure 4.10), we can satisfy $f_\alpha \cdot O_\alpha^{l16} = 1$, $O_{l19}^{23} = 1$ and $O_{l23}^{29} = 1$ separately to obtain a test vector,

Lines *l6, l14, l15* and *l26* must then be equal to 0, 1, 1 and 1 respectively to propagate the fault to the primary output 129. Note that among these lines, only *l6* is a primary input. To set *l14* equal to 1, we need to choose a path from the BDD of line *l14* that leads to terminal node *1*. Let *l1=0* and *l2=1* ($l14 = l1 \oplus l2$). 126 can be equal to 1, when *l22=1*, *l12=1* and *l13 =0*. To set *l22=0*, we choose, for example, *l10=1* and *l9=0*. In this way, a test vector is found hierarchically.

```
Hierachical_test_Gen(P)
  P: a primary output
{
   - Find the maximal supergates
   For each fault f (l0 s-a-v) such that l0 belongs to P
   {
      set all variables to X

      if(v==0) /* l0 s-a-0 */
      {
         if((S=f_l0 · O_l0^L1) == 0)
           return; /* The fault cannot be tested through P*/
        Satisfy(S, 1); /* Let SG(L1) be the supergate to which l0 belong */
      }
      else
      {
         f((S=f̄_l0 · O_l0^L1) == 0)   return;
        Satisfy(S, 1);
      }
      /* The fault is propagated to the output of the supergate SG(L1) */
      i=1;
      repeat
        {
         if( O_Li^Li+1 == 0) return /* a fault at Li cannot be propagated to Li+1 */

         satisfy( O_Li^Li+1, 1); /* Let SG(L_i+1) be the supergate to which L_i belong
         i++;
        }
      until(L_i+1 = P) /* we reach the supergate of the primary output P */
   }
}
```

Figure 4.11 Test generation for a line *l0 s-a-v*

```
Satisfy(F, val)
F is a boolean function represented by a BDD.
val is a logic value (0 or 1)
{
  -choosc assignment to the lines on which F depends to have F=val;
  /* Any path from the BDD of F that leads to terminal node v is a solution*/
  for each variable V on which F depends /* supergate input */
   {
    if the logic value taken by V   ≠   X and V does not correspond to a primary input
     {
      /* V corresponds to a supergate output /*/
      if(the logic value taken by V is 0)
       satisfy(f_l, 0); /* f_l is the boolean function at line l*/
      else
       satisfy(f_l, 1);
     }
   }
}
```

Figure 4.12 Recursive procedure that chooses an assignment hierarchically to the PIs to have F=val

Given a primary output P, the algorithm in Figure 4.11 tries to generate a test vector to any fault *l0 s-a-v* where *l0* belongs to the circuit of P. First, by using algorithm 1, the circuit is decomposed and the maximal supergates are found. The important property of a supergate is that all its inputs are mutually independent. This will enable hierarchical generation of test vectors for each undetected fault. In this way, the BDD of any line of the circuit will contain nodes corresponding to either PIs or supergate output and any assignment to these lines that creates a path in the BDD leading to a terminal node does not lead to signal conflicts (does not affect two different logic values on the same line of a subcircuit). First it propagates the fault to the output of the supergate of the line. Next, the algorithm propagates the fault to the output of the subsequent supergate by choosing an appropriate assignment to the inputs of this supergate making the output of supergate 1 observable at the output of supergate 2). We proceed in this way until PO P is reached.

The *Satisfy*() procedure, shown in Figure 4.12, is a recursive procedure, taking a boolean function and a logic value v as parameters. This procedure tries to find an assignment to formula variables that makes $F = v$. If a variable A of F is not a primary input, in other words, an output of a supergate and must be equal to, for example, 0, then the procedure *Satisfy*() will be called by the following parameters: The boolean function of A and 0. It proceeds in this way until all the boolean functions are satisfied (the primary inputs are reached).

4.6 Experimental Results

Our algorithm is implemented in the C programming language. The results were obtained by running the algorithm on a SUN/4 workstation.

After fault-collapsing, two phases of test pattern generation follow: random and algorithmic. The first phase of test pattern generation is the random phase, using the fault simulator *fsim* [14]. In this way, we generate patterns for the easily tested faults (generally 80% to 99% of the total faults).

Table 4.1 Result obtained by running the fault simulator *fsim*

Circuit	Total faults	Collapsed faults	Detected faults	Number of vectors	CPU(s)
c432	864	524	502	57	0.333
c499	998	758	713	54	0.333
c880	1660	942	878	72	0.550
c1355	2710	1574	1405	60	0.783
c1908	3816	1879	1500	45	1.25
c2670	5340	2239	2239	89	1.3
c3540	7080	3428	2850	97	2.76
c5315	10630	5350	5085	106	3.07
c7552	15104	7550	6785	131	5

Table 4.1 shows the results obtained by running the parallel random fault simulator *fsim*. It shows, for each circuit, the number of faults (total faults), the number of collapsed faults, the number of detected faults, the number of test vectors and the CPU time. When *fsim* completes the second phase, algorithmic test pattern generation begins by calling our program. During the algorithmic pattern generation phase, each pattern generated is simulated (using a single fault propagation simulator) so that any faults detected by the new pattern may be removed from the fault list.

Table 4.2 Test generation using our hierarchical approach

Circuit	Maximum no. of subcircuits	Remaining faults	Redundant faults	No. of vectors	Total no. of vectors A.C	CPU(s)
c432	11	22	4	13	52	10.01
c499	1	45	8	22	56	204
c880	9	64	0	32	71	6.2
c1355	1	169	8	52	89	218
c1908	1	379	9	159	138	83
c2670	14	508	117	96	117	520
c3540	2	578	137	145	190	8000
c5315	3	265	59	86	147	55.13
c7552	6	765	131	155	240	90

Table 4.2 shows the results obtained by running our algorithm on the ISCAS'85 benchmark circuits, and includes the number of subcircuits (maximal supergates). Since we study the primary outputs separately, we decompose the circuit of each primary output. As a result, a different number of subcircuits can be found for each primary output. The number given in Table 4.2, column 2, corresponds to the maximum number of subcircuits found. Note that when the circuit cannot be decomposed the number of supergates will be

1. In column 3, we find the number of faults remaining after random vector generation, followed by the number of redundant faults found, the number of test vectors generated algorithmically, the compacted set of test vectors (which includes the vectors found by *fsim*) and the CPU time.

The input variables take a different ordering for each primary output. It is interesting to note that all the ISCAS'85 benchmark [7] circuits have been studied except c6288. This example is not studied since it is easy to test using the random fault simulator *fsim*.This is a 32-bit multiplier for which it has been proven that the ROBDD must always be of exponential size for some outputs, even for optimum input ordering.

Table 4.3 Test generation for some sequential circuits

Circuit	Maximum no. of subcircuits	Collapsed faults	No. of faults P.R	No. of vectors A.C	CPU(s)
s298	2	308	0	40	0.3
s420	6	455	0	75	1.6
s444	6	474	14	38	0.5
s713	6	581	38	38	5.07
s832	5	870	14	120	2.37
s953	9	1079	0	100	3
s1488	6	1486	0	131	3.67
s1494	6	1506	12	13	4.02
s5378	10	4503	40	280	4.02
s15850	15	11725	389	560	3000
s35932	6	39094	3984	75	930

Table 4.3 shows results for some ISCAS'89 benchmark circuits [12]. ISCAS'89 were processed under the full-scan assumption.

Table 4.4 gives the results obtained by running the algorithm TSUNAMI [13] on a DECstation 5000/200 on the ISCAS'85 and ISCAS'89 benchmarks. Table 4.5 gives the results of the performance of the algorithm A2, presented in [25], on the ISCAS'85 circuits. A2 was run on a SUN 4 workstation.

Table 4.4 Results of TSUNAMI [13]

Circuit	Faults	Gen time	Patterns	Compress time
c432	524	14.5	39	2.5
c499	758	187.0	78	178.1
c880	942	48.0	25	5.5
c1355	1524	560.6	111	493.7
c1908	1879	1307.8	121	92.3
c5315	5350	177.5	71	136.9
s298	308	0.2	28	0.1
s420	430	1.1	46	0.6
s444	470	0.5	30	0.2
s713	581	1.8	25	0.6
s832	870	2.8	104	1.3
s953	1079	3.8	103	1.5
s1480	1480	7.9	114	2.1
s1494	1506	8.2	114	2.1
s5378	4603	105.4	112	46.4
s15850	11725	892.6	1582	4873.5
s35932	39094	5089.3	5235	4622.5

It is very interesting to note that a good comparison concerning the CPU time between our algorithm and the other algebraic algorithms based on BDDs cannot be done because:

1) The input-variables ordering is different for each algorithm. This will affect the CPU time, since in the three approaches boolean functions must be manipulated to generate test vectors.

2) The set of undetected faults to be considered by each algorithm is not the same for each circuit.

3) In [13] and in [25] the authors use a single ordering for all the outputs of the circuit,and in these cases each fault is studied once. In our case (in this version of the algorithm) we use an separate ordering for each PO. This means that in many cases a fault will have to be considered repeately (once for each PO to which the fault can be propagated) and so many BDDs will have to be recomputed according to the ordering of the PO considered.

Table 4.5 Results of algorithm A2 [25]

Circuit	#faults	Forwards pass time	Test Gen time	Total
c432	6	1.2	1.7	2.9
c499	23	12.0	273	285.0
c1908	39	7.31	105.6	112.9
c1355	20	20.9	42.8	63.7
c3540	41	37.6	33.1	70.7
c5315	40	20.1	23.5	43.6

Based on the results shown above, we can conclude that with our approach more complex circuits can be studied, for example c7552 and c2670, than with existing methods based on BDD representation, [13] and [25] (see tables 4 and 5 respectively). In addition, the time spent for each circuit in generating test vectors and finding the redundant faults is short. It is interesting to note that by using a single ordering for the variables,

CPU time can be enormously reduced, but the number of circuits to be studied will be also reduced. Using circuit decomposition (supergates), test vector generation time is improved and more complex circuits can be studied, since smaller BDDs will be manipulated.

4.7 Conclusion

We have presented an algebraic technique for generating tests for single stuck-at faults in combinational circuits. The concept of dominance is used for identifying the set of maximal supergates. This enables hierarchical generation of test vectors for stuck-at faults and accelerates test vector generation, since the size of the BDDs to be manipulated is reduced.

By using variable ordering for each primary output, test vectors have been generated for a larger set of networks than existing methods based on BDDs. In addition, we have studied many benchmark circuits that cannot be studied using a single ordering for all primary outputs.

Using the observability concept, once a fault related to a gate is studied (a test is generated or the fault is declared redundant), all the other undetected faults related to that gate can be studied with almost no additional effort. For all the circuits analyzed, the algorithm is faster than previous algebraic methods. It is also very efficient for hard-to-detect faults and redundant faults.

Experimental results with ISCAS'85 and ISCAS'89 benchmarks show that this approach is well-suited to test generation for faults that are very hard to detect or redundant faults in a circuit.

CHAPITRE 5

Modélisation et génération automatique de vecteurs de test pour les circuits mixtes

Introduction

Le présent chapitre est basé sur un papier de conférence intitulé "*Graph Modeling and an Automatic Test Vector Generator for Mixed-Signal Circuits*" qui a été publié dans *European Design &Test Conference* en Mars 1995.

Nous présenterons dans ce chapitre une nouvelle technique de génération de vecteurs de test pour les circuits mixtes (contenant des blocs analogiques et numériques). Cette méthode est basée sur une modélisation d'un circuit mixte, au complet, par un seul graphe.

Le circuit mixte est modélisé de telle façon que des informations peuvent être transmises facilement d'un bloc à un autre. La technique proposée prend en considération les contraintes imposées par chacun des blocs d'un circuit mixte. Nous démontrerons que, généralement, aucune modification n'est requise pour tester ce type de circuits.

Nous utiliserons la méthode décrite dans [37] pour la modélisation adéquate d'un bloc analogique d'un circuit mixte par un graphe. Le graphe est construit en se basant sur les calculs de sensibilité. Les composants, ainsi que les paramètres, du circuit analogique

sont représentés par des noeuds. Un arc valué existe entre un noeud correspondant à un élément (composant) et un autre correspondant à un paramètre si et seulement si le paramètre est sensible à la variation du composant. Le poids de l'arc représente la variation du composant pour laquelle la valeur du paramètre sort de la zone permise.

Dans cette approche, nous utiliserons la représentation au niveau portes logiques pour modéliser les blocs numériques. Nous avons trouvé qu'en utilisant ces deux types de modélisation, nous pouvons différencier un comportement normal d'un mauvais comportement du circuit mixte considéré. Par conséquent, nous pouvons automatiser la génération de vecteurs de test pour les circuits mixtes.

Pour générer des vecteurs de test pour les blocs numériques, nous utiliserons une approche algébrique. Ainsi, les contraintes, ou les conditions imposées par les blocs analogiques peuvent être considérées facilement en manipulant des fonctions logiques. La même approche algébrique sera utilisée aussi pour propager d'un bloc à un autre une erreur résultante d'une panne dans un bloc analogique.

Nous avons démontré dans [57] que l'automatisation de la génération de vecteurs de test est possible. Nous avons trouvé qu'il y a une certaine analogie à exploiter entre la génération de test pour une panne dans le circuit analogique et celle pour une panne dans le circuit numérique.

Dans notre approche, nous étudions le cas des pannes simples (c'est-à-dire, il existe au plus une panne dans le circuit). Pour un circuit analogique, une panne est tout simplement un élément qui se trouve en dehors de la région de tolérance permise. Pour tester une telle panne, nous commençons d'abord par mesurer le paramètre le plus sensible à cette

panne. Pour un circuit numérique, une panne est une ligne qui est collée à 0 ou à 1. Pour la tester, il faut d'abord l'activer et propager l'erreur résultante à travers la porte logique dont l'observabilité est la plus grande.

Dans ce chapitre, nous supposerons que nous n'avons aucun accès aux sorties analogiques du circuit mixte. Celles-ci sont connectées à des blocs de conversion qui sont, à leur tour, connectés aux entrées des circuits numériques. Donc, pour détecter l'existence d'une panne dans le circuit analogique, l'erreur résultante doit être propagée à travers le bloc numérique. Pour propager l'erreur, on a besoin d'imposer au moins une valeur composée à une sortie du bloc de conversion. Une valeur composée contient l'information sur le comportement d'un circuit défectueux et d'un circuit normal. Par exemple, une valeur composée 1/0, ou D, à une ligne quelconque *l* implique que la ligne *l* est égale à 1 lorsque le circuit est normal (sans défectuosité) et 0 lorsque le circuit est défectueux.

Pour rendre la propagation de l'erreur une tâche plus facile, nous supposons que les lignes du circuit dépendent d'une autre variable. Cette variable n'a rien à voir avec le circuit numérique. Elle sera utilisée seulement pour représenter une valeur composée. Ainsi, nous décrivons, en même temps, le comportement du circuit normal et celui d'un circuit défectueux.

Etant donné qu'on n'a pas d'accès aux sorties du bloc analogique, pour tester une panne dans ce bloc, il faut donc propager la panne à travers le bloc de conversion et le bloc numérique. Or, dans certains cas, on n'est pas capable de rendre observable des lignes du circuit de conversion. Par conséquent, il se peut que certains éléments du circuit analogique ne peuvent être testés que pour des tolérances beaucoup plus grande que cel-

les permises. Dans ce cas, une panne testable lorsque le circuit analogique est considéré seul peut devenir non testable lorsque le circuit fait partie d'un circuit mixte.

Notons aussi, qu'à cause des contraintes imposées par le circuit analogique, des pannes testables du circuit numérique, lorsqu'il est considéré seul, peuvent devenir non-testables lorsque le circuit numérique fait partie d'un circuit mixte.

Le travail présenté dans ce chapitre a été fait avec la collaboration de Naim Ben Hamida qui a été responsable du test des blocs analogiques d'un circuit mixte. Quant à moi, Béchir Ayari, j'était responsable de la génération de vecteurs de test pour les blocs numériques. J'était responsable aussi de la propagation des erreurs dues à des pannes dans un des blocs analogiques à travers les blocs numériques.

Nous avons uni nos efforts pour automatiser la génération de vecteurs de test pour les circuits mixte. Ensemble, nous avons travailler sur l'adaptation des deux méthodes et sur la modélisation de l'interface entre les blocs.

Graph Modeling and an Automatic Test Vector Generator for Mixed-Signal Circuits

Bechir Ayari, Naim BenHamida and Bozena Kaminska

Ecole Polytechnique of the University of Montreal

P.O. Box 6079, Station "Centre-ville", Montreal, PQ, Canada, H3C 3A7

Abstract

Mixed-signal circuit testing is known to be a very difficult task. This is due to the difficulty of controlling the digital signal from the analog outputs, observing the analog outputs in the digital circuit, controlling the analog circuit from the digital outputs and observing the digital signals in the analog circuit. As a solution to these problems, we propose first a graph modeling for mixed-signal circuits. Second, based on this modeling, an automatic test vector generator is presented to perform functional testing. In this paper, only the case of an analog block followed by a digital block is studied. To test the analog part, the best parameters to be measured are selected. For digital circuit testing, the single stuck-at fault model is considered and an algebraic method is used. The conditions imposed by each block, called constraints, are taken into account during test vector generation. The experimental results (simulation and discrete realization) show the efficiency of the automatic test generation technique.

5.1 Introduction

Recent improvements in fabrication technology have made possible the realization of reliable integrated circuits (ICs) containing both analog and digital functions on the same silicon chip. The problem of testing these circuits is, however, much more complicated than that of testing purely digital or analog ICs. Different techniques have evolved for the two types of circuit which are difficult to integrate into a single testing solution. Digital circuit testing consists in checking that the pattern of 1's and 0's at the outputs corresponds to the pattern expected. Analog testing consists in measuring, for example, gain, bandwidth, distortion, impedance, noise, etc.

Because the problems mentioned above are difficult to handle, generically specific solutions for standard mixed circuits such as CODECs [29], ISDN [30] and A/D converters [31] have been proposed. The other alternative is the use of Design for Testability (DFT) rules, that basically partition the mixed circuit into analog and digital sections so that the input and output signals of each section can be accessed. One possible implementation of this technique is the use of some analog and digital multiplexers for controllability/observability purposes [32] and [33]. A possible implementation is the use of the mixed-signal test bus standard IEEE P1149.4 [34].

Note, however, that isolating the analog and the digital part of a circuit, during test mode has its associated problems. Both chip area and the number of I/O pins will have to increase to accommodate the test requirements. These increases may be quite excessive in circuits containing many analog and digital blocks. Another problem with these techniques is that we are not sure about the quality of the test since the circuit is not tested in

its normal mode. Thus, some errors which are due to the interaction between digital and analog blocks may not be tested.

Alani et al. [35] presented a steady-state-response test generation method for mixed-signal integrated circuits. This test generation algorithm reduces test overhead by grouping the analog and digital blocks. This technique considers catastrophic faults only. Also, the circuit should be modified before being tested by this technique.

Other alternative for testing mixed-signal circuits without partitioning them into digital and analog blocks is presented in [36]. The mixed-signal ICs are tested as an entity by the application of specific forms of pulsed excitations and the capture of the device's composite transient response via externally presented circuit nodes. The transient response of the circuit is captured using a high-speed A/D converter and then transferred to a computer for subsequent signal analysis.

In this paper, a new modeling for mixed-signal ICs is presented. Based on this modeling, a test method for these kind of circuits is also presented. In our case, the circuit is considered as an entity, so there is no need for circuit partitioning into analog and digital blocks. Also, there is no need of high speed A/D converter or circuit modifications using DFT techniques. The proposed test generation technique consists of functional testing for the analog parts and test vector generation with constraints for the digital parts. This technique allows the test of every block under the conditions imposed by the other blocks. For purpose of consistency only the case of *analog-digital* circuits is considered.

The rest of the paper is organized as follows: Section 5.2 deals with mixed-circuit modeling. In Section 5.3 the testing procedures for the analog circuit is reviewed. The test

vector generation with constraints (for digital circuit) and fault propagation from one block to another are also presented in Section 5.3. The automation of test vector generation is shown is Section 5.4. Some experimental and validation results are discussed in Section 5.5 and Section 5.6, respectively. Finally, a conclusion is given in Section 5.7.

5.2 Mixed-circuit modeling

Generally, a mixed-signal circuit is composed of three building blocks: a digital block, an analog block and a conversion block. Mixed-signal circuits can be classified into two categories from which any mixed-signal circuit can be built. In the first category we find *analog-digital* circuits, which are made up of an analog block, an A/D converter and a digital block. In this case, an analog signal has to be treated in a digital environment (Figure 5.1). In the second category, we find *digital-analog* circuits, which are generally made up of a digital block, a D/A converter and an analog block. From these two categories any other mixed circuit can be built.

In this paper, we will focus our attention only on *analog-digital* circuits. In Figure 5.1 we have an *-analog-digital* circuit where the only accesses are its primary inputs and primary outputs. In other words, we assume that we do not have any access to the analog output and some of the primary inputs of the digital circuit.

In order to test these kind of circuits as an entity, without any partition, a modeling method is needed for each block of the circuit. In addition, modeling methods must be compatible to automatize test vector generation and to easily propagate testability informations.

For a digital circuit, we have found that the gate-level representation is the most appropriate modeling. In this way, the structural informations about the circuit can be easily represented by a graph. Based on the structural informations in the graph, functional informations can be computed easily by boolean function manipulations using the Ordered Binary Decision Diagram (OBDD) representation. Such informations are very useful for test vector generation and error propagations. In addition, the use of an algebraic approach allows to take into consideration the constraints that can be imposed by each block of a mixed-signal circuit.

For an analog circuit, a graph modeling is also needed. The schematic representation of an analog circuit cannot be used for this purpose, since testability informations cannot be easily extracted. In addition, this kind of modeling will not allow the automatization of test vector generation. The modeling needed must overcome the nonlinearity and reduce the complexity of the relation between analog inputs and outputs. We have found that the analog circuit modeling technique presented in [37], based on sensitivity computation, is the best candidate. It will be used to construct a graph representing the relation between components (elements) and parameters. More details about this modeling is given in Section 5.2.1

The A/D converter circuit is itself a mixed-signal circuit, since it is generally composed of an analog block and a digital block. In this case, the two modeling methods, described before, can be merged to model the conversion circuit. Section 5.3 deals with graph modeling of the conversion block.

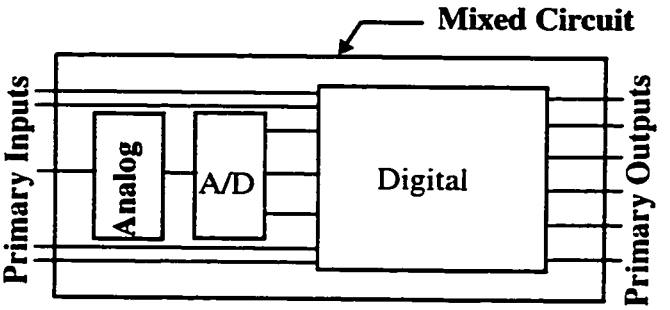


Figure 5.1 Analog-digital circuit

5.2.1 Analog circuit modeling

The analog circuit modeling, that we are interested in, must efficiently represent the behaviors of a fault-free and a faulty circuits. Before introducing this modeling, let us introduce the kind of faults that can occur in an analog circuit.

Faults in analog circuits can be categorized as catastrophic or parametric (soft). Catastrophic faults are open and short circuits caused by sudden and large variations in components [37]. Parametric or soft faults are defined by the circuit's functionality.

For an analog circuit, the concern is which parameters to select for testing and the accuracy to which they should be tested in order to detect the variations in the faulty components. It is known that the circuit's parameters depend on the circuit's elements, since they represent the functions realized by the circuit. For example, the gain of an amplifier is a function of the set of transistor characteristics, resistors, capacitances, etc. Then, if for example, all the elements on which the gain depends are fault-free, the amplifier's gain will be also fault-free. Furthermore, if all the parameters of the amplifier are fault-free, then the amplification circuit will be fault-free.

In our opinion, to test an analog circuit, we should test all the elements that make up the circuit. In other words, we have to test if all the elements are in their tolerance boxes. Then any element whose value is out of its tolerance box is considered faulty. In a faulty circuit, we suppose that only one element is faulty and all the others are fault-free.

Using the modeling technique presented in [37], a weighted bipartite graph representing the relation between components (elements) and parameters is constructed. An edge between an element and a parameter exists if the parameter is sensitive to the element. The edge weight represents the sensitivity value, or the maximum tolerances that can be tolerated. In other words, the deviation of the element that makes the parameter, connected to the same element, be out of its tolerance box.

We distinguish two kinds of element's tolerances: The first is the tolerance of a fault-free element, which is taken from the circuit's data sheets. The second tolerance is the faulty element tolerance, the worst element tolerance computed using Equation 5.1. Using this equation, for every parameter T_k depending on the faulty element x_i, the maximum value of the tolerance of element x_i is computed, assuming that the other elements are fault-free and take the maximum values of their tolerance boxes. The value computed using Equation 5.1 represents the variation in x_i that can be seen by observing the parameter T_k in the worst case. In other words, if the deviation exceeds this tolerance, the considered parameter will be out of its tolerance box.

$$\left(\frac{\Delta x_i}{x_i}\right)_{Max} = \frac{\left|\frac{\Delta T_k}{T_k}\right| + \sum_{j=1}^{i-1}\left|S_{x_j}^{T_k}\right|\left|\frac{\Delta x_j}{x_j}\right| + \sum_{j=i+1}^{M}\left|S_{x_j}^{T_k}\right|\left|\frac{\Delta x_j}{x_j}\right|}{S_{x_i}^{T_k}} \quad (5.1)$$

Where:

$S_{x_i}^{T_k}$: the différential sensibilty of the parameter T_k with respect to to the element x_i.

ΔT_k: the deviation of the output parameter with respect to its nominal value T_k.

T_k: the nominal value of the of the k^{th} parameter.

x_j: the nominal value of the j^{th} component.

Δx_j: represents the deviation of the component value with respect to its nominal value x_j

Let us take the second order band-pass filter of Figure 5.2 to illustrate the modeling procedure for analog circuits. This filter is composed of eight elements {R1, R2, R3, R4, Rg, Rd, C1, C2} and five parameters: A1, the center-frequency gain, A2, the gain, f_0, the center frequency, f_{c1}, low cut-off frequency, and f_{c2}, the high cut-off frequency.

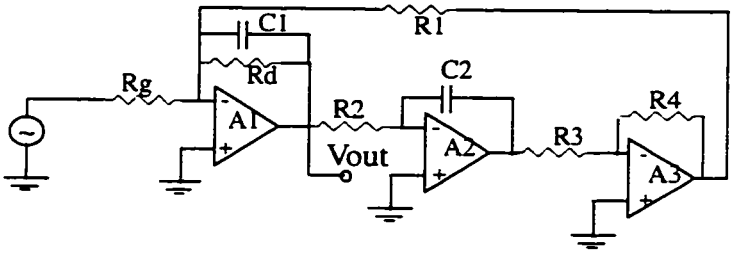


Figure 5.2 A second order band-pass filter

To obtain the connectivity matrix, shown in Table 5.1, we have assumed that parameter tolerances are equal to 5%. We have assumed also that an element is considered fault-free if its tolerance is less than or equal to 5%. The worst-case deviation is computed for all the elements and each parameter, using Equation 5.1.

Table 5.1 represents the connectivity matrix corresponding to the band pass filter of Figure 5.2. It represents the graph modeling of the analog circuit. This graph represents the relation between the elements and the parameters. A shaded cell in the table indicates that the corresponding parameter is not sensitive to the corresponding element. In this case, the element and the parameter are not connected in the graph.

Table 5.1 The connectivity matrix of the band pass filter of Figure 5.2

	$\frac{\Delta R1}{R1}$	$\frac{\Delta R1}{R1}$	$\frac{\Delta R2}{R2}$	$\frac{\Delta R3}{R3}$	$\frac{\Delta R4}{R4}$	$\frac{\Delta Rg}{Rg}$	$\frac{\Delta C1}{C1}$	$\frac{\Delta C2}{C2}$
$\frac{\Delta A1}{A1}$					10.1	9.9		
$\frac{\Delta A2}{A2}$	28.9	28.9	28.9	28.9	176	176	27	28.9
$\frac{\Delta f_0}{f_0}$	36.3	36.3	36.3	32.2			36.3	36.3

Table 5.1 The connectivity matrix of the band pass filter of Figure 5.2

$\frac{\Delta f_{c1}}{f_{c1}}$	44.5	44.5	44.5	37.2	37.2		40.4	38.3
$\frac{\Delta f_{c2}}{f_{c2}}$	30.5	30.5	30.5	40.1	40.1		28.7	30.5

5.2.2 Conversion-circuit modeling

Since this paper deals with only analog-digital circuits, in this section, we will focus our attention only on modeling of analog to digital conversion circuits. An analog-to-digital conversion circuit can be an A/D converter or any other comparison circuit, like the one presented in Figure 5.3, [36].

Any A/D converter is composed of two parts, a comparison circuit and a decoding logic circuit. The decoding circuit is digital and the comparison circuit is made up of a number of comparators and reference voltages. The comparison structure is an analog circuit. So, it can be modeled by a graph, as described in Section 5.2.1.

As an example, let us consider the comparison circuit shown in Figure 5.3, which is composed of three resistors and two comparators. Since the reference voltages Vt1 and Vt2 are parts of an analog circuit, the relations between Vt1 and Vt2 and the elements (resistances) can be modeled by a graph. Figure 5.4 shows the graph model of the comparison circuit of Figure 5.3. This kind of modeling tells us, for example, that a fault in the element Rc1 can be tested by fixing the conversion block input *Vin* and by measuring either Vt1 or Vt2 (Since we have an edge from Rc1 to Vt1 and another from Rc1 and Vt2). Note that this kind of modeling fits well with digital circuit modeling.

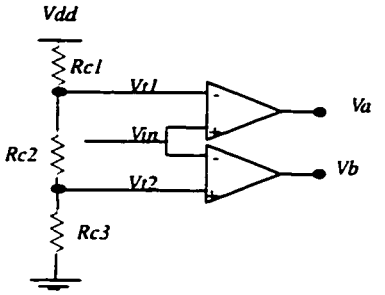


Figure 5.3 Comparison circuit

Using this kind of modeling, a faulty element in the conversion circuit or in the analog block can be propagated through the comparators by choosing an appropriate value for *Vin* (Figure 5.3 and Figure 5.4)

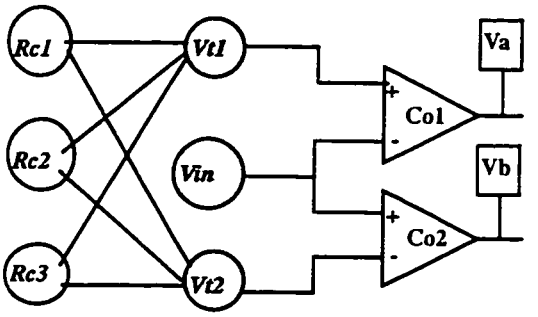


Figure 5.4 Graph model of the comparison circuit

5.2.3 Modeling of the whole mixed-signal circuit

Let us consider the mixed signal circuit shown in Figure 5.5 which is composed of three blocks: the filter discussed in section 2, Figure 5.2, the conversion block of Figure 5.3 and a digital circuit. The corresponding graph model is shown in Figure 5.6. Note that, for analog circuit modeling purposes, another node is needed. This node, called a translation block (TB), see Figure 5.6. Each cell of TB has two inputs, the analog primary input,

API, and one of the considered parameters. The output of the translation block is an analog signal. Given the element values, the parameter to be measured and the analog input, the translation block gives us the corresponding analog signal output of the fault-free circuit. For any faulty element, the corresponding faulty output can be deduced using the informations contained in the analog modeling-graph.

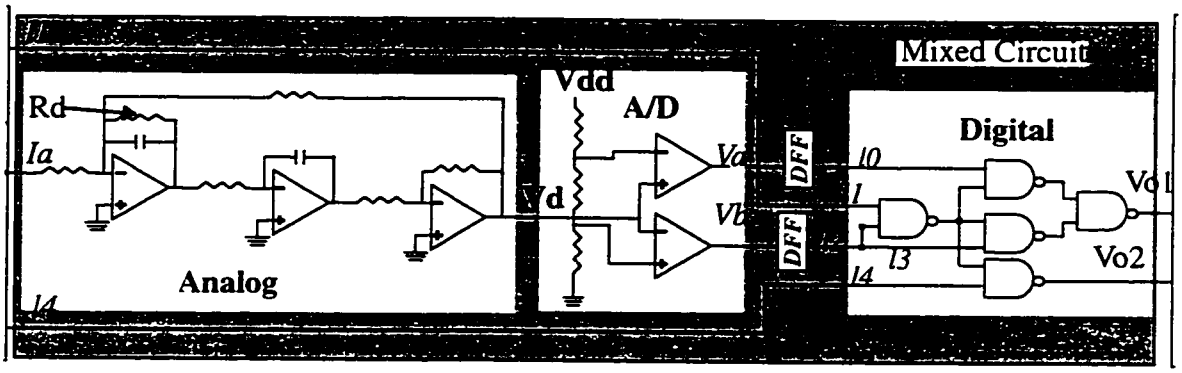


Figure 5.5 Mixed-signal circuit

Using this modeling for an *analog-digital* circuit, an error in the analog block can be easily propagated to one primary output of the circuit, by choosing an appropriate assigning the digital primary inputs and the analog primary input.

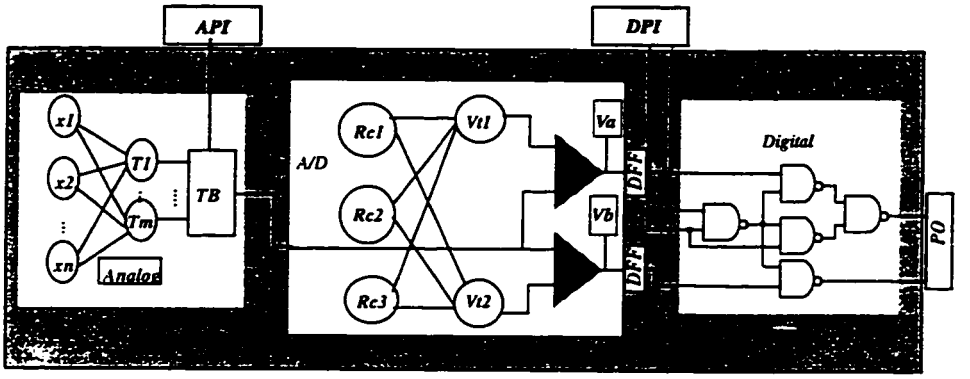


Figure 5.6 Graph modeling of the analog digital circuit of Figure 5.5.

5.3 *Analog-digital* circuit testing

Testing a mixed-signal circuit means finding out if the circuit meets some of its specifications. These specifications can be element and parameter tolerances that have to be respected, or a fault-free circuit whenever a fault model is considered. So, the problem of mixed-signal testing is to convert the parameter specifications from the analog domain to the digital domain. In other words, we have to:

1) Control the digital signals from the analog inputs and,

2) Propagate the analog signals through the digital part.

Our test strategy for a mixed circuit is based on the analog and digital circuit modelings presented in Section 5.2. The testing technique adopted for each block of the circuit must take into consideration the conditions imposed by the other blocks. The element testing technique of analog circuits, based on the modeling presented in Section 5.2.1, is used for testing the analog block of the mixed circuit [37]. For digital circuit testing, our auto-

matic test vector generation technique [41], based on BDD and boolean difference, is adapted in order to include the constraints imposed by the analog block.

Given an *analog-digital* circuit, as the one shown in Figure 5.5, the digital block should be tested under the conditions imposed by the analog block, since some of its inputs are controlled by the same analog signals. For the analog block, since its output cannot be directly accessed, the resulting error of a faulty element in this block should be propagated through the digital block to the mixed-signal primary outputs.

5.3.1 Test vector generation for the digital block

For any digital circuit, the fundamental steps in generating a test vector for a fault *l s-a-v* (v=0 or v=1) are, first, to activate (excite) the fault, and, second, to propagate the resulting error to a primary output (PO). But, generally, in a mixed circuit, many lines of the digital part can be controlled by the same analog signal. This will create dependency between some of the inputs of the digital circuit. Consequently, another problem will rise: many assignments to some digital circuit lines cannot be obtained by controlling the analog signals of the circuit. As a result, these constraints must be taken into account while activating a fault or propagating an error in the digital block. Then, the assignments to circuit lines must satisfy the conditions imposed by the analog block.

If an enumeration technique is used for test vector generation. It tries to find an assignment to activate the fault and after that see if the fault can be propagated to a PO. Using such technique, we will find that in many cases, a great deal of backtracking will be required. This method becomes impractical, particularly when the circuit contains many redundant faults. In addition, the constraints cannot be added easily to this testing strategy.

The question that rises is: How can we take into consideration constraints brought about by the analog part?

The approach presented in [41] describes an algebraic method based on OBDD representation for digital circuit testing. To generate a test vector to a given stuck-at fault, the approach computes the boolean difference between the boolean functions computed for the same output of the fault-free and faulty circuits. It uses separate ordering for the primary outputs to study more complex circuits and the dominator technique to accelerate test vector generation. Using this approach, the redundant faults are studied as easily as the detectable faults.

This method can be adapted for testing a digital block in a mixed-signal circuit, by adding the conditions to satisfy the generated test vector. In our case, the assignments representing the allowed assignments are represented by a boolean function called *Fc* (the constraint function). *Fc* is a sum of product terms represented by an OBDD. Each product term, a path leading to 1 in the OBDD, represents an allowed assignment to the lines depending on the analog part. Then, any assignment that makes *Fc* equal to 1 can be obtained by controlling the analog signal. Note that if all the assignments are allowed *Fc* will be equal to 1. As a result, there is no constraint to satisfy while generating test vectors. Note that *Fc* represents the set of assignments that can be obtained by controlling the analog input and the boolean difference represent the set of test vectors. Then, the intersection of both sets, obtained by ANDing the two corresponding functions, represents the test vector that satisfies the constraints. This way, we obtain directly the set of test vectors that activate the fault, propagate the error and satisfy the constraints at the same time. Then, for a mixed circuit, the test vector (assignments to PIs) must satisfy 3 conditions: 1)

activate the fault, 2) propagate the fault to a primary output, and 3) satisfy the constraints imposed by the analog part.

$$E = \{(PI_0,PI_1,\ldots,PI_{n-1}) \mid S = f_l \cdot F_c \cdot PO_{l=D} = D \text{ or } S = f_l \cdot F_c \cdot PO_{l=D} = \overline{D}\}$$

represents the set of test vectors for the fault l s-a-0 that satisfy the constraints. $PO_{l=D}$ represents the BDD of the primary output PO with the assumption that l is a primary input called D. In this way, any assignment that leads to node D allows the propagation of an error in line l to the primary output PO. Note that we AND the corresponding function by f_l to activate the fault l s-a-0, and the resulting function with Fc to take into consideration the constraints.

Similarly,

$$E = \{(PI_0,PI_1,\ldots,PI_{n-1}) \mid S = \bar{f}_l \cdot F_c \cdot PO_{l=D} = D \text{ or } S = \bar{f}_l \cdot F_c \cdot PO_{l=D} = \overline{D}$$

represents the set of test vectors for the fault l s-at-1 that satisfies the constraints.

Example 1:

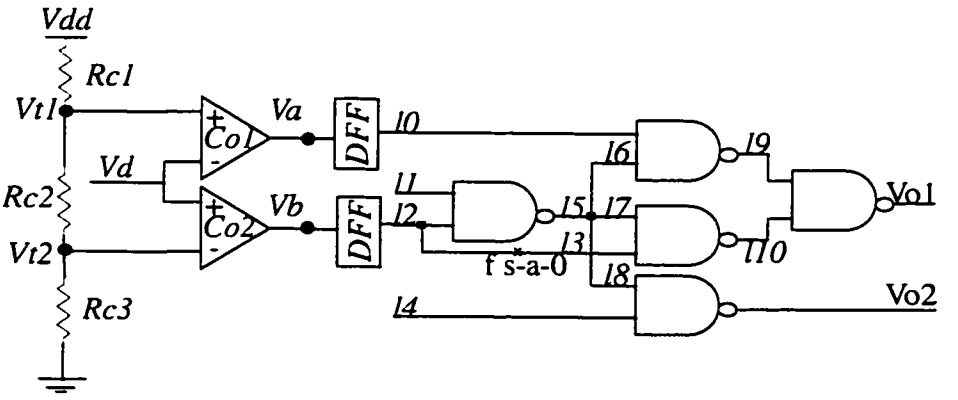


Figure 5.7 A two-output circuit with the fault *l3 s-a-0*

Consider the circuit of Figure 5.7, where the lines *l0* and *l2* of the digital circuit are connected to Va and Vb respectively. In this case, the logic value of *l0* and that of *l2* will be controlled by the same analog input. As a result, these two lines become dependent. Note that we cannot control these lines to zero at the same time. Suppose also that we have to generate a test vector for line *l3* s-a-0. Then we have $f_{l3} = l2$, $Vol_{l3=D} = (\overline{l2} + \overline{l1}) \cdot (l0 + D)$ and $f_{l3} \cdot Vol_{l3=D} = \overline{l1} \cdot l2 \cdot (l0 + D)$. For this example, $F_c = l0 + l2$ (*l0=0* and *l2=0* cannot be obtained). Consequently, $S = f_{l3} \cdot F_c \cdot Vol_{l3=D} = \overline{l1} \cdot l2 \cdot (l0 + D)$. Note that $S = D$ only when *l0*, *l1* and *l2* are equal to 0, 0 and 1 respectively. Then the test vector $\{l0, l1, l2, l4\} = \{0, 0, 1, X\}$ tests the fault and satisfies the constraints.

When considered alone, the digital circuit of Figure 5.7 is found to be fully testable, which means that 100% coverage can be obtained. But, when this circuit is a part of the mixed circuit the fault coverage has changed. The dependencies introduced by the analog part have an effect on the fault coverage of the digital block. This is the *constraint* under which the digital circuit is tested. For the circuit of Figure 5.7, we have found that 2 of the 18 uncollapsed single stuck-at faults considered become undetectable when both circuits are connected. Note that the faults *l0* s-a-1 and *l3* s-a-1 cannot be tested, since lines *l0* and *l2* cannot be controlled to 0 and 0 respectively.

5.3.2 Analog circuit testing

In our approach, the problem of analog testing is to find a set of parameters that guarantee the maximum possible coverage of all elements. This testing procedure can be used also for testing the faults inside the OpAmps by choosing the appropriate model of the

OpAmp. The models of OpAmps presented in [42] and [43], that consider all possible faults, can be used for this purpose.

Finding a test set for the elements of the circuit consists in finding a set of parameters to be measured that guarantee maximum fault coverage of the elements. The maximum fault coverage of an element is defined as the minimum element deviation that can be observed by measuring one parameter. The element coverage is defined as the minimum element deviation that can be observed at one primary output parameter at least. To quantify the coverage of an element, we should compute the relative deviation of the faulty element, and, where the other elements are fault-free, their tolerances are taken from the circuit's technical notes.

If we apply a DC signal in the analog primary input and the comparator inputs are not affected with noise, all the PIs of the digital part controlled by this analog input will have a logic value equal to 0 or to 1. Suppose that we have to generate test vectors for the analog parts of the mixed circuit in Figure 5.5. In this circuit, we have assumed that we have only one analog input. Suppose that we have a faulty component in the analog part of the mixed circuit. In other words, the element deviates from its specification tolerance. Then, to find a test vector, we must first activate the fault and then try to propagate the resulting error through the conversion and the digital blocks.

To activate the fault, the parameter that allows testing the least tolerance of the element is chosen. In order to propagate the fault through the conversion block, the signal to be applied in the analog primary input must force at least one output of the A/D converter to have a different value in the fault-free circuit and in the faulty one. In other words, first, the faulty element should force the measured parameter to be out of its tolerance, other-

wise the fault is not testable and, second, the parameter deviation should force at least one of the outputs of A/D converter to have a value different from the fault-free value. Suppose that a parameter is considered faulty, due to an element deviation, when its deviation exceeds 5%. The analog input should be chosen such that we have an output of the A/D converter that has different values when the parameter deviation is inside the tolerance box [-5%,5%] and when it is outside the tolerance box.

When an output of the analog circuit cannot be accessed, the value taken by this output must be observed at the digital outputs. Then, an error in the analog block has to be propagated through at least one comparator of the conversion block and through the digital block, as indicated in Figure 5.8. Consequently, to activate a fault in the analog block, we must impose different logic values for the fault-free and faulty circuits at the output Vd (Figure 5.8) of at least one comparator of the conversion block.

To test any element/parameter of the analog circuit of Figure 5.8, the amplitude A and the frequency f of the applied signal must be chosen appropriately. Table 5.2 gives, for each parameter to test, the corresponding amplitude and frequency of the signal which should be applied to the analog circuit in order to have a composite value at one comparator output. The techniques for choosing the amplitudes and frequencies (depending on the cut-off frequencies of the analog circuit) are described in [37]. To test a parameter T deviation $\frac{\Delta T}{T}$, two vectors are needed, one to test the upper bound of a parameter deviation and the other to test the lower bound, Table 5.2. A parameter T is considered fault-free if its deviation is inside the tolerance box [-x,+x] ($\frac{\Delta T}{T} \in [-x, \ +x]$), otherwise it is faulty. Table 5.3 gives the notation of the used parameters.

Table 5.2 Test set of an analog circuit parameters

Param. (T)	Test	Input signal		$\frac{\Delta T}{T} < -x$ (faulty)		$-x < \frac{\Delta T}{T} < x$ (fault-free)		$\frac{\Delta T}{T} > x$ (faulty)		comp. value
		f	A	Va	Vd	Va	Vd	Va	Vd	
A_{DC}	$A_{DC} > (1+x) \cdot A_{DCn}$	0	$\frac{V_{ref}}{(1+x)A_n}$	$<V_{ref}$	0	$<V_{ref}$	0	$>V_{ref}$	1	$\overline{D}$
	$A_{DC} < (1-x) \cdot A_{DCn}$	0	$\frac{V_{ref}}{(1-x)A_n}$	$<V_{ref}$	0	$>V_{ref}$	1	$>V_{ref}$	1	D
A_{AC}	$A_{AC} > (1+x) \cdot A_{ACn}$	$f>0$	$\frac{V_{ref}}{(1+x)A_f}$	$<V_{ref}$	0	$<V_{ref}$	0	$>V_{ref}$	1	$\overline{D}$
	$A_{AC} < (1-x) \cdot A_{ACn}$	$f>0$	$\frac{V_{ref}}{(1-x)A_f}$	$<V_{ref}$	0	$>V_{ref}$	1	$>V_{ref}$	1	D
f_{lcf}	$f_{lcf} > (1+x) \cdot f_{lcfn}$	f_{lcfn}	$\frac{V_{ref}}{(1-y)A_{f_{lcfn}}}$	$>V_{ref}$	1	$>V_{ref}$	1	$<V_{ref}$	0	D
	$f_{lcf} < (1-x) \cdot f_{lcfn}$	f_{lcfn}	$\frac{V_{ref}}{(1+y)A_{f_{lcfn}}}$	$>V_{ref}$	1	$<V_{ref}$	0	$<V_{ref}$	0	$\overline{D}$
f_{hcf}	$f_{hcf} > (1+x) \cdot f_{hcfn}$	f_{hcfn}	$\frac{V_{ref}}{(1+y)A_{f_{hcfn}}}$	$<V_{ref}$		$<V_{ref}$	0	$>V_{ref}$	1	$\overline{D}$
	$f_{hcf} > (1+x) \cdot f_{hcfn}$	f_{hcfn}	$\frac{V_{ref}}{(1+y)A_{f_{hcfn}}}$	$<V_{ref}$	0	$>V_{ref}$	1	$>V_{ref}$	1	D

Table 5.3 Definitions of the notations used in Table 5.2

A_{AC}	AC gain of the analog circuit
A_{ACn}	nominal AC gain of the analog circuit
A_f	AC gain of the analog circuit when the frequency of the signal is f
A_{DC}	DC gain of the analog circuit
A_{DCn}	nominal DC gain of the analog circuit
f_{lcf}	cut-off frequency of a low-pass filter
f_{lcfn}	nominal cut-off frequency of a low-pass filter
$A_{f_{lcfn}}$	AC gain corresponding to the frequency f_{lcfn}
f_{hcf}	cut-off frequency of a high-pass filter
f_{hcfn}	nominal cut-off frequency of a high-pass filter
$A_{f_{hcfn}}$	AC gain corresponding to the frequency f_{hcfn}
y	The deviation seen in the gain, when the frequency deviates by x% from its nominal value
V_{ref}	a voltage reference from the conversion block

Let us explain, for example, how to test a deviation on the parameter f_{lcf} (cut-off frequency of a low pass filter) of the analog circuit. Suppose that f_{lcf} deviates from the nominal value by more than x[%]. Suppose also that a deviation of x[%] in the frequency causes a deviation of y[%] in the gain of the analog circuit. In this case, according to Table 5.2, the amplitude B and the frequency f of the analog signal to be applied are respectively $\frac{V_{ref}}{(1-y)A_{f_{lcfn}}}$ and f_{lcfn}. Since we have a low-pass filter, when f exceeds its nominal value by more than x%, A_f would decrease by more than y%. In other words, when f > (1+x)f_{lcfn}, $A_f < (1-y)A_{f_{lcf}}$. In this case, Va= $A_f \cdot B\sin(2\pi f)$ is always less than

V_{ref}. In the other case, Va is greater than V_{ref} in a period of time Tp. Tp depends on the nominal frequency and its deviation. To propagate the fault, we use *composite* logic values of the form v/v_f, where v and v_f are values of the same signal in N (fault-free network) and N_f (the network with fault f). In this case, the logic value of the digital circuit lines can be 0,1, D, $\overline{D}$, or equals a boolean function. To propagate the fault, we simply traverse the circuit beginning from the lines at which the logic value is D or $\overline{D}$ to the POs of the mixed circuit, see procedure *propagate_through_digital*() of Figure 5.11. We compute the OBDD for each line traversed. If this OBDD does not contain the node D, the fault cannot be propagated through this line, so we try another line. In the opposite case, we continue our traversed until we reach a PO. If the OBDD generated contains D, the fault can be tested, and a test vector is generated by simply choosing a path in the OBDD leading to D.

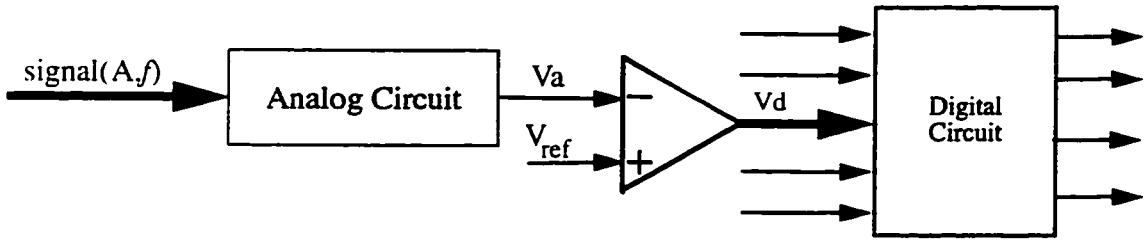


Figure 5.8 *Analog-digital* circuit

As in the digital circuit, a fault in the analog part can only be tested if two conditions are satisfied: 1) The fault is activated, and 2) The fault is propagated to a primary output. To activate the analog fault, we must choose the amplitude and the frequency of the analog signal properly. To propagate the fault to a primary output, we choose an appropriate assignment to the primary inputs of the mixed circuit.

Let us take the circuit in Figure 5.5 which is composed of three sub-circuits:

1) Analog sub-circuit, a second order filter that have one input *Ia* and one output Vd connected to the input of the A/D converter.

2) An A/D converter whose two outputs are connected to the inputs *l0* and *l2* of the digital circuit.

3) The digital circuit which has two external primary inputs, *l1* and *l4*, and two other inputs, *l0* and *l2*, which are connected to the outputs of the A/D converter.

Suppose now that we are interested in testing the deviation of a parameter in the analog block. To test such a fault, we need to activate the fault. In other words, we must choose an analog signal (an amplitude and a frequency) to apply at the input *Ia* that creates a change in the behavior of at least one of the A/D converter outputs. After activating the fault, we have D or $\overline{D}$ at one or many outputs of the A/D converter and we must propagate this fault to a primary output of the mixed circuit. Then, an appropriate assignment to the primary inputs *l1* and *l4* must be chosen in order to have D or $\overline{D}$ in at least one of the primary outputs.

Suppose that the element Rd of the second order band-pass filter (Figure 5.5) is faulty. According to Table 5.1, a deviation in Rd of less than 9.9% cannot be tested. When the faulty element deviation is greater or equal to 9.9%, this fault can be tested by measuring the amplification gain A1. So if A1 deviation exceeds 5% then the fault can be observed at the output of the analog circuit, otherwise it cannot. To test whether or not A1 deviation is inside its tolerance box [-5%,5%], the upper and lower bounds of the box should be tested. The first alternative is to observe the fault effect at the output of the com-

parator Co1, Figure 5.5. So, Vb will have different values in the faulty and fault-free circuit.

Let us begin by testing the lower bound of the tolerance box. To do so, a sine wave having a frequency of 10khz and an amplitude B should be applied at the analog input of the mixed circuit. B should be chosen in order that the amplitude of Vd is greater than or equal to V_{t2}, when the deviation is greater than or equal to -5%, and when the faulty element forces A1 to decrease from its nominal value by more than 5%,Vd will be less than V_{t2}. Consequently, $B = \frac{V_{t2}}{0.95A1}$ will force Vb to switch from 1 to 0. So, in the fault-free case Vb is 1 and in the faulty case it is 0, which means that in the line *l0* of the digital circuit we will have a composite value equal to D and *l2* = $\overline{D}$. The same reasoning will be applied in order to test the upper bound of A1 except that $B = \frac{V_{t2}}{1.05A1}$.

In order to find an assignment to the input lines *l*1 and *l4*, we first generate the OBDD of the output Vo1 with *l0*=D and *l2*= $\overline{D}$ and we check whether or not there exists a node corresponding to D in this OBDD. If Vo1 OBDD contains D, the fault can be propagated to this output. Otherwise, we generate the OBDD of one of the other outputs.

The OBDDs of Vo1 and Vo2 constructed with *l0*=D and *l2* = $\overline{D}$ are represented in Figure 5.9. Note that we can propagate the fault at either of the two outputs, since in the corresponding OBDDs we have a node D. Then, when we set *l*1=1, the fault is propagated to Vo1, and when we set *l*1=1 and *l*4=1, the fault is propagated to both outputs Vo1 and Vo2.

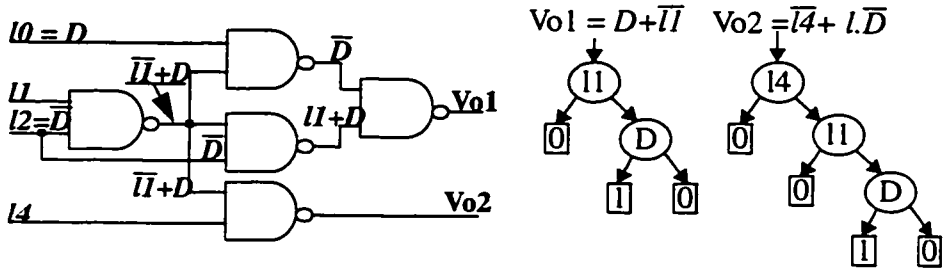


Figure 5.9 Propagation of the error caused by the analog fault to an output of the mixed circuit

5.4 Automation of Test Vector Generation

To automatize test vector generation, the graph modelings of analog and digital blocks are needed. For cut-frequency testing, the deviations in the amplitudes of the output signal, that correspond to the maximum deviations tolerated, are needed. The other information needed, in addition to the modeling graph of the analog circuit, is how the blocks of the mixed-signal circuit are connected. These informations are needed for constraint function computation during test vector generation of the digital block.

If all these informations are given to our software tool, test vectors are generated automatically for all the blocks of the mixed-signal circuit.

We have shown in [41] that, by studying the primary outputs of a digital circuit separately, test vector can be generated for more complex circuits. In addition, if the circuit can be decomposed the time spent for test vector generation can be reduced and the size of OBDDs will be manageable. For generating test vectors for the digital part the only information we need is the constraint function that depend on how the blocks are connected.

The procedure shown in Figure 5.10 is used to generate test vectors for all the elements of the analog circuit. For each element, the parameter that is the most sensitive to a deviation in the element is taken first. Using Table 5.2, this procedure will find an analog signal that will activate the fault. In other words, it chooses an amplitude and a frequency that sets D or $\overline{D}$ at one of the primary outputs of the conversion block. Next, we try to propagate the error to one primary output of the mixed-signal circuit by calling the procedure shown in Figure 5.11. When "all the possibilities are studied", in other words, when all the cases that allow to have D or $\overline{D}$ at one of the primary outputs of the conversion block have been tried, and the fault cannot be propagated through the digital block, it is impossible to test the fault by measuring the deviation induced on T. Then, we look for another parameter from the parameter set ($\{T\}$) of x_i. When all the parameters of the element x_i have been studied without success, any deviation in this element cannot be seen at any primary output of the mixed circuit. Here, we have supposed that all the POs of the mixed circuit come from the digital block.

```
Test_analog_elements(x_i)
    x_i: an element of the analog circuit;
{
    T: a list of parameters related to x_i;
    for(T = x_i->parameter; T != NULL AND x_i is not tested; T = T->next){
        Repeat{
            - Choose an amplitude that sets D or D̄ at one
              of the primary outputs of the conversion block
            If(Propagate_through_digittal() == True)
              x_i is tested
        }until all the possibilities are studied or x_i is tested
    }

}
```

Figure 5.10 Procedure to test all the analog-circuit elements

After activating a fault in the analog circuit, some of the comparators will have composite values that correspond to the fault-free and faulty circuit. The procedure of Figure 5.11 is first called in order to propagate the resulting error through the digital block. It tries to propagate the fault beginning from lines in which we have a composite value.

```
Propagate_through_digital()
{
    * Find the composite values (fault-free/faulty value)
      at each output of the conversion block;
    * set 0, 1, D, D̄ to the primary inputs controlled by the
      primary outputs of the conversion block;
    for (i =0; i < number_of_PIs; i++){
        if (i is connected to the conversion block AND value[i] = D or D̄){
            Propagate_fault_through(i);
        if the fault is propagated
            return (True);
        }
    }
    return(False);
}
```

Figure 5.11 Procedure *Propagate_through_digital()*

The recursive procedure *Propagate_fault_through()*, shown in Figure 5.11, enables propagation of the resulting error through the digital part. It takes a line through which it tries to propagate the fault. It constructs the OBDD at this line and checks whether or not the OBDD contains node D. If it does, it continues its traversal. When the line reached is a primary output and the OBDD constructed contains node D, the fault is propagated by choosing an appropriate assignment to the inputs of the digital block. Note that no propagation will occur when the constructed OBDD does not contain the node D.

```
Propagate_fault_through(i)
 i: index of a gate in the digital circuit
{
    if the fault is not propagated {
        bdd_gate = const_bdd(i);
        if (D   ∈    bdd_gate){
            if (the output of gate[i] is a primary output)
              the fault is propagated;
            else{
             for (fanout_gate = gate[i]->fanout; fanout_gate!= NULL;
                 fanout_gate =fanout_gate ->next)
              Propagate_fault_through(fanout_gate->index);
             }
            }
        }
    }
}
```

Figure 5.12 The Procedure Propagate_fault_through(i)

5.5 Experimental results

In order to show the effectiveness of the proposed test generation technique, the results are given: (1) when every block is considered alone and (2) when it is considered as a part of the mixed-signal circuit. The experimental results are given for two examples: the first one is presented in Figure 5.5. The second example is composed of a fifth order low-pass chebyshev filter, a conversion circuit made of 15 comparators and 16 resistors, and a digital circuit, one of the ISCAS85 benchmark circuit [7].

Test vectors are generated for the mixed circuit of Figure 5.5. In case 1 the analog, digital and conversion blocks are considered separately. Thus, we have a direct access to PIs and POs of every block As shown in the example of Figure 5.5, in order to have the

best coverage of all the elements in the analog circuit, when single fault is considered, the parameters (*T*) A1 and A2 have to be tested. If the parameter deviation is less than 5% then the element error is less than the computed element deviation (E.D). The same E.D. can be tested for the analog circuit in case 1 and case 2. This is due to the fact that there is at least one comparator output which can be tested for stuck-at-0 and stuck-at-1 faults, and at which the analog faults can be activated. The A/D conversion testing is similar to the analog testing since we propose to test the elements (Rc1,Rc2,Rc3) of the circuit by measuring the voltage references (vt1,vt2).

Table 5.4 Test results for the *analog-digital* circuit of Figure 5.5

Case 1			**Case 2**		
Analog block alone			***Analog block is a part of the mixed-signal Circuit***		
T	***E***	***E.D.***	***T***	***E***	***D***
A1	Rd	10.1	A1	Rd	10.1
	Rg	9.9		Rg	9.9
A2	R1,R2,R3,R4,C2	28.9	A2	R1,R2,R3,R4,C2	28.9
	C1	27		C1	27
Digital block alone			***Digital block is a part of the mixed-signal circuit***		
NCF	***NUF***		***NCF***	***NUF***	
18	0		18	2	
A/D alone			***A/D is a part of the mixed-signal circuit***		
T	***E***	***ED[%]***	***T***	***E***	***ED[%]***
Vt1	R1	8	Vt1	R1	8
Vt2	R2	18	Vt2	R2	18
	R3	6.5		R3	6.5

Since it is possible to propagate D and $\overline{D}$, in line l1 and line l2, to at least one primary output, a fault in the conversion-circuit element can be easily propagated through the digital layer. This is the reason why the element deviation (E.D.) is the same in case 1 and case 2. The digital circuit is fully testable when it is considered alone. In fact, when we have a direct access to the digital block PIs the number of tested faults is equal to the number of collapsed faults (NCF). But when the digital block is a part of the mixed circuit, the number of undetected faults (NUF) is 2. Two faults became untestable due to the constraints imposed by the analog block.

Example 3

In this example, the analog block is a fifth-order-chebychev filter, the conversion circuit is a comparison circuit made of 15 comparators and 16 resistors. The output of the filter feeds the comparators and the 15 outputs of the comparators are directly related to digital circuit inputs. Since the digital circuit may have more than 15 inputs, the selection of the digital inputs, that are controlled by the comparators, is performed randomly. With this conversion circuit, the digital circuit became more difficult to test because of the constraints imposed by the comparators.

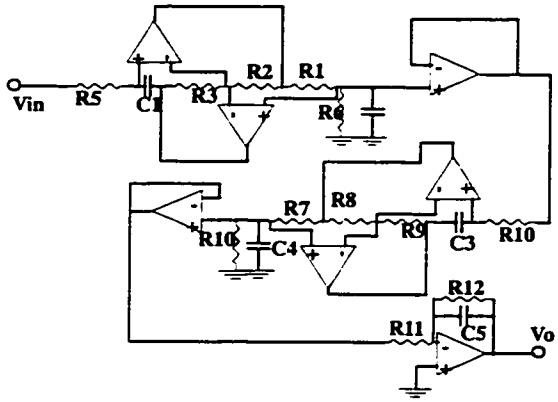


Figure 5.13 Fifth-order-chebychev filter

In case 1, we consider the circuit to be tested is the analog block (Chebychev filter, Figure 5.13). In this case we have access to the POs and PIs of the analog circuit. In case 2, the analog block is a part of the mixed-signal circuit. In this case, the PI of the analog circuit is the only access point to the circuit, and the analog circuit output should be observed at the digital POs.

The Chebychev filter is composed of three blocks. In case 1, since we have access to the output of the third block only, some elements cannot be tested accurately. This is the case of the element R5. In fact a deviation less than 113% in R5 may not be tested if the worst case is consideredTable 5.5). This is due to the lack of accessibility to the analog circuit. In case 2, when the output of the third block of the chebychev filter is connected to the conversion block, the elements are tested by the same accuracy as in case 1, Table 5.5. This is due to the fact that it is possible to propagate D and $\overline{D}$, to at least one comparator output.

Table 5.5 Test results for the fifth-order-chebychev filter.

Case 1			Case 2		
Analog block alone			*Analog block is a part of the mixed-signal circuit*		
T	***E***	***ED[%]***	***T***	***E***	***ED[%]***
Adc	R8	25.9	Adc	R8	25.9
	R7	24.4		R7	24.4
fc	R2	45.7	fc	R2	45.7
	R3	19.5		R3	19.5
	C1	9.32		C1	9.32
A1	R4	11.8	A1	R4	11.8
	C2	14.6		C3	14.6
A2	R9, C5	14.5	A2	R9,C5	14.5
A3	R5	113	A3	R5	113
	C3	41.7		C3	41.7
	R4	31.3		R4	31.3
A4	R6	56.1	A4	R6	56.1
A5	R1	49.6	A5	R1	49.6

For the digital block, the results obtained for some benchmark circuits are shown in Table 5.6. This table shows, for each circuit, the number of primary inputs (#PI), number of primary outputs(PO), the number of collapsed faults. It shows also, for each circuit, the number of untestable faults, number of vectors and the CPU time in both cases (with and without constraints). The constraints are imposed by the conversion block composed of 15 comparators and 15 reference voltages. In the presented results, 15 PIs are chosen randomly from each circuit and considered to be controlled by the analog block.

Table 5.6 Test vector Generation, with and without constraints, for some benchmark circuits

Circuit	#PI	#PO	Collap. Faults	Without constraints			With constraints		
				#Untestable Faults	#vect.	CPU [s]	#Untestable Faults	#vect.	CPU [s]
c432	36	7	524	4	52	220	11	56	937
c499	41	32	758	8	56	318	8	65	1230
c880	60	26	942	0	71	10	12	63	61
c1355	41	32	1574	8	89	680	12	104	1574
c1908	33	25	1979	9	138	1025	81	119	2200

According to Table 5.6, when constraints are added to the test vector generator, we note that the fault coverage and the time spent for test vector generation are affected. An increase in the number of untestable faults is noted for all the circuits but C499. In the first case, when we have no constraints on the PIs of a circuit, a random test vector generator can be used to accelerate test vector generation. In the second case, a random test pattern can be simulated only if it satisfies the constraints imposed by the analog block of the mixed circuit. For this reason we have chosen to generate all the test vectors deterministically.

Now let us see from which comparator the analog faults can be activated and propagated. To test if the deviation on a parameter exceeds its tolerance or not, the amplitude and a frequency of the analog signal is chosen according to Table 5.2. Since the amplitude depends on a reference voltage, and there are 15 comparators in the conversion block, the fault can be observed at the 15 POs of the comparators. In this table, we have studied the case when only one primary output of the conversion block behaves differently in a faulty and in a fault-free circuit. It is shown in Table 5.7 that, for the circuit C499, if a deviation

in the amplitude is less than -5%, then analog faults cannot be propagated through 4 comparators. This means that the reference voltages connected to those comparators cannot be tested. All the reference voltages should be tested in order to have the best coverage of all the resistors of this block. This is possible in case 1, but in case 2, when the conversion block is considered as a part of the mixed-circuit, some reference voltage cannot be tested. As a result, this will have a direct effect on the conversion-block-element deviations that can be tested. We note also, that for the same circuit, any deviation greater than 5% in the gain can be propagated through any comparator to a primary output.

Table 5.7 Propagation of faulty parameters through comparators

Circuit	Total Number of PI	PIs from conversion block	Number of PIs through which we cannot propagate the fault $A < (1-x)A_n$	Number of primary inputs through which we cannot propagate the fault $A > (1+x)A_n$	CPU
c432	36	15	1	1	150
c499	41	15	4	0	41.2
c880	60	15	1	0	3.5
c1355	41	15	2	0	58.3
c1908	33	15	1	1	39.5

According to Table 5.8, the minimum deviation on R5 (62%) can be tested by observing the value taken by Vt5. But, when the analog circuit is connected to the c432 circuit, many dependencies are created between some of the digital inputs. As a consequence, we note that R5 cannot be tested for a deviation of 62%, Table 5.9. According to the same table, we note also that R5 can be tested only for a deviation greater than 71%, in the worst case, through the comparator connected to Vt6 (Table 5.9).

Table 5.8 Conversion-circuit element coverage when its input and outputs are directly accessed

Case1															
T	Vt1	Vt2	Vt3	Vt4	Vt5	Vt6	Vt7	Vt8	Vt9	Vt10	Vt11	Vt12	Vt13	Vt14	Vt15
E	R1	R2	R3	R4	R5	R6	R7	R8,R9	R10	R11	R12	R13	R14	R15	R16
ED[%]	15	31	41	51	62	71	81	91	77	64	52	40	29	17	6

Table 5.9 Conversion-block element coverage when it is a part of a mixed circuit.

Comparators connected to c432															
T	Vt1	Vt2	Vt3	Vt4	Vt5	Vt6	Vt7	Vt8	Vt9	Vt10	Vt11	Vt12	Vt13	Vt14	Vt15
E	R1	R2	R3	R4		R6,**R5**	R7	R8,R9	R10	R11	R12	R13	R14	R15	R16
ED[%]	15	31	41	51		71	81	91	77	64	52	40	29	17	6
Comparators connected to *c499*															
E		**R1**,R2	R3	R4		R6,**R5**	R7	R8,R9	R10, **R11**		R12	R13	R14	R15,**R16**	
ED[%]		31	41	51		71	81	91	77		52	40	29	17	
Comparators connected to c880															
E	Vt1	R2	R3	R4	R5	R6	R7	R8,R9	R10	R11	R12	R13	R14, **R15**		R16
ED[%]	15	31	41	51	62	71	81	91	77	64	52	40	29		6
Comparators connected to c1355															
E	Vt1	R2	R3	R4	R5		R7,**R6**	R8, R9	R10	R11	R12	R13	R14	R15,**R16**	
ED[%]	15	31	41	51	62		81	91	77	64	52	40	29	17	
Comparators connected to c1908															
E	Vt1	R2	R3	R4	R5	R6		R8,R9, **R7**	R10	R11	R12	R13	R14	R15	R16
ED[%]	15	31	41	51	62	71		91	77	64	52	40	29	17	6

Table 5.8 gives the results of the conversion-block-elements testing in case 1, whereas Table 5.9 gives the results for the conversion-block-element testing in case 2. Table 5.8 shows the element deviation that can be tested when the digital block is one of the following benchmark circuits: c432, c499, c880, c1355, c1908. The dashed cells in Table 5.9 tells us that the reference voltage cannot be tested. Any element can be tested by

more than one reference voltage. From the testable reference voltages, we choose the one that tests the minimum element deviation.

5.6 Result validation

In order to validate the proposed technique for mixed circuit testing a discrete realization of analog-digital circuit is performed. The analog circuit is the state variable filter, the digital circuit is a 4-bit adder and the conversion circuit is an 8-bit A/D converter, Figure 5.14. The experimental results are shown for every block considered as alone and when they are a part of a mixed circuit.

With an incremental sensitivity analysis and the computation of the worst case component deviation, we found that to guarantee the maximum coverage of all the components, the performances $\{A_{1dc}, A_{2dc}, A_{3dc}, A_{3'dc}, A1_{10khz}, A2_{10khz}, f_{h1}\}$ should be measured. These performances are selected among $\{A_{1dc}, A_{2dc}, A_{3dc}, A_{3'dc}, A_{max1}, A_{max2}, A_{max3}, A1_{10khz}, A2_{10khz}, A3_{10khz}, f_1, f_2, f_3, f_{h1}, f_{h2}, f_{h3}, f_{l1}, f_{l2}, f_{l3}\}$ where:

A_{idc}: DC gain computed at output Vi, for i=1 to 3.

$A_{3'dc}$: DC gain computed at output V3 when Vin is below a threshold voltag

A_{imax}: Maximum AC gain computed at output Vi, for i=1 to 3.

f_i: frequency of the maximum AC gain computed at output Vi, for i=1 to 3.

f_{li}: Low cut-off frequency computed at output Vi, for i=1 to 3.

f_{hi}: High cut-off frequency computed at output Vi, for i=1 to 3.

Some experimental results are provided for the state variable filter using discrete components realization. The operation amplifier used in this realization is the uA741. The output signal of the non-faulty filter is first measured, then, a fault is injected in the filter and the output signal is observed. The values of the faults (component deviations) are given in Table 5.10. In this experiment, we suppose that we have only one faulty element. All the possible faults were injected, and we found that the computed worst case component deviation forces the measured performance deviation (MPD) to be out of its tolerance box ([-5%,+5%]).

Furthermore, the performance deviation obtained experimentally is far beyond the tolerance box value. This means that a fault which is even less than the computed component deviation (CD) can be detected, Table 5.10. Note that it also means that our computation is very pessimistic since we consider the worst case, and this does not occur frequently. Then, we have observed that every fault greater than the CD is easily detected at the output of the digital block.

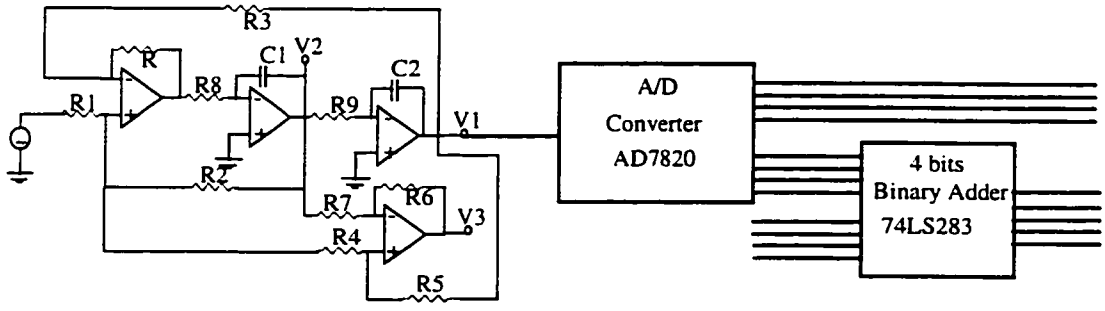


Figure 5.14 A mixed circuit composed of a state variable filter, A/D converter and a 4-bit adder

Table 5.10 shows the performances to be measured (T), components (C), the deviation at which they are tested (CD) and the measured parameter deviation (MPD). It is

obvious that all the component deviations can be detected, since this deviation forces the measured parameter to be out of its tolerance box. The components are tested with the same accuracy when the analog circuit is considered alone and when it is a part of the mixed circuit of Figure 5.14, since all the considered faults can be propagated through the digital circuit.

Table 5.10 Test results for the state variable filter.

T	*C*	*CD[%]*	*MPD[%]*
A1dc	R3	21	13
A2dc	R1	13	9.1
	R2	8	5
A3	R6	20	5.5
	R7	21	6.5
A3'dc	R4	23	21
	R5	23	28
A1	R9	24.5	44
	C2	24.5	31
A2	R8	26	50
	C1	26	33
fh1	R	20	14

For the digital circuit, we can inject a fault (stuck-at 0 or stuck-at 1) only at the input of the 4-bit adder. For example to test a s-a-0 fault at one of its inputs, the net is shorted to the ground. The test applied is generated using our algorithm (test generation with constraints). We have found that any fault at one input of the adder can be tested by controlling the external inputs (analog and digital inputs).

5.7 Conclusion

In this paper we have proved that a class of mixed circuit can be tested as entity without modification. The analog inputs can be easily activated and the fault can be propa-

gated through the digital block using an algebraic method based on BDDs. For digital circuit testing, the current implementation uses the single stuck-at fault model. The dependency introduced by the connection of the blocks (digital, analog and conversion) makes test vector generation harder for test generators based on gate-level representation. We used an algebraic method based on OBDD representation that allows us to efficiently manipulate boolean functions. The constraints imposed by the analog block, are taken into account during test vector generation. Simulation results as well as the practical validation confirm the efficiency of the proposed method.

CHAPITRE 6

Génération aléatoire et déterministe de vecteurs de test pour les circuits séquentiels

Introduction

Le présent chapitre est basé sur un papier de conférence intitulé "*Random and Deterministic Test Vector Generation for Sequential Circuits*" qui a été soumis à *IEEE Transactions on Circuits And Systems*.

La génération de vecteurs de test pour les circuits séquentiels est une tâche très difficile à faire. En effet, jusqu'à maintenant, les générateurs de vecteurs de test pour les circuits séquentiels ne peuvent être appliqués qu'à un ensemble restreint de circuits avec un état "reset" ou à des petits circuits.

Nous présenterons dans ce chapitre une génération mixte de vecteurs de test, ou si vous voulez, une génération pseudo-aléatoire suivie d'une génération déterministe. Notre approche a deux avantages: le premier est que le nombre de pannes à considérer par notre générateur déterministe sera réduit, et par conséquent la génération de vecteurs de test sera accélérée. Le deuxième avantage c'est que l'ensemble de vecteurs de test, générés de façon pseudo-aléatoire, sera utilisé comme une séquence d'initialisation qui mettra le cir-

cuit dans un état partiellement connu. En utilisant cette approche, la génération déterministe sera, généralement, plus facile à faire.

Nous présenterons une méthode algébrique, c'est-à-dire basée sur la manipulation de fonctions logiques, pour générer des vecteurs de test. Elle est basée sur la représentation par un diagramme de décision binaires des fonctions logiques et le modèle itératif d'un circuit séquentiel. Une génération aléatoire de vecteurs de test sera appliquée pour tester les pannes facilement testables. Ainsi la liste des pannes, à considérer par la génération déterministe sera réduite.

Notre méthode utilise les mesures de testabilité présentées dans [11] pour estimer le nombre maximum de *time-frames* à considérer dans le modèle itératif. Elles seront utilisées aussi pour la compaction des vecteurs pseudo-aléatoires et l'ordonnancement des variables de l'entrée d'un circuit séquentiel.

Nous connaissons que la génération pseudo-aléatoire de vecteurs de test pour les circuits combinatoires a été utilisée efficacement pour minimiser le nombre de pannes à considérer dans la génération déterministe de vecteurs de test. Elle permet la détection des pannes facilement testables. Généralement, la génération déterministe de vecteurs de test ne sera appliquée que pour tester des pannes difficilement testables et détecter les pannes redondantes.

Jusqu'à maintenant, la génération pseudo-aléatoire n'a pas été utilisée par les générateurs de vecteurs de test pour les circuits séquentiels. Ceci est dû au fait qu'un noeud séquentiel nécessite, généralement, une séquence de vecteurs pour être mis dans un état connu.

Pour un circuit combinatoire, seuls les vecteurs pseudo-aléatoires qui détectent des pannes seront conservés. Mais pour un circuit séquentiel, on doit garder des séquences de vecteurs.

Notons qu'aucune méthode n'a été développée pour déterminer quels sont les vecteurs qu'il faut garder parmi les vecteurs aléatoires générés, sans affecter la couverture de pannes. En se basant sur des mesures de testabilité, Naim *et al.* [10] ont montré comment sélectionner les séquences de vecteurs qui détectent des pannes. Ils ont montré aussi que, dans la plupart des cas, la couverture de pannes ne sera pas affectée. Dans ce cas, il sera possible d'utiliser la génération pseudo-aléatoire comme une première étape de la génération de vecteurs de test pour les circuits séquentiels.

Le travail présenté dans ce papier a été fait aussi avec la collaboration de Naim Ben Hamida qui a été responsable du calcul des mesures de testabilité et de la génération aléatoire de vecteurs de test pour les cicuits séquentiels. Quant à moi, Béchir Ayari, j'était responsable de la génération déterministe de vecteurs de test qui inclue l'ordonnancement des variables des BDDs ainsi que la manipulation de fonctions logiques.

L'article est organisé comme suit: dans la section 2 nous présenterons des mesures de testabilité pour les circuits séquentiels. Nous discuterons leur utilité dans la compaction de vecteurs de test pseudo-aléatoires. La section 3 montre notre approche de génération des BDD pour les circuits séquentiels. La technique de génération est décrite à la section 4. Nous concluons à la section 5.

Random and Deterministic Test Vector Generation for Sequential Circuits

Ayari Bechir, BenHamida Naim and Bozena Kaminska

Department of Electrical and Computer Engineering

Ecole Polytechnique de Montréal

P.O. Box 6079, Station "Centre-Ville", Montréal, Canada (H3C 3A7)

Abstract

Sequential ATPG (Automatic Test Pattern Generation) is viewed today as a very desirable CAD tool. To date, the size and complexity of circuits for which sequential ATPG could be applied is limited. In this paper, a sequential ATPG which is based on mixed (random and deterministic) generation is proposed. In order to reduce the fault list, a set of random vectors is applied first. From this set, and based on probabilistic testability measures (TMs), only those vectors which detect faults are conserved. Deterministic test pattern generation, which is based on boolean function manipulations using ROBDD (reduced ordered binary decision diagram) representation is applied to the remaining untested faults. For sequential ROBDD construction, the iterative model of sequential circuits is used. The length of the iterative model and the sequential variable ordering are determined using the same probabilistic TMs. For deterministic test vector generation, the ROBDD of the faulty as well as the fault-free circuits is constructed in order to have a test

vector (an assignment to the inputs for which the output of the faulty circuit is different from that of the fault-free circuit). The efficiency of our test vector generation technique is also illustrated through extensive simulation results.

6.1 Introduction

Performing sequential test pattern generation is well known to be a very hard task. In fact up to now, sequential ATPGs could only be applied either to a restricted set of circuits with a reset state [45], [46] and [47] or it is limited to small circuits [48]. Most sequential ATPGs are time-consuming, giving poor fault coverage for medium-sized circuits. Our research deals with the generation of test patterns for sequential circuits without the assumption of a reset state. It is based on a single observation test time strategy rather than on a multiple observation strategy [49].

Pseudo-random vector generation for combinational circuits has been efficiently used in ATPG tools. In fact, some faults are easily detected by pseudo-random vectors, and random-vector generation is not time-consuming. Up to now, pseudo-random generation has not been used in sequential ATPG tools. This is mainly due to the fact that a sequential node generally needs, more than one vector to be set in a known state.

For combinational circuits, it is clear that only those pseudo-random vectors which detect faults are to be conserved. However, for sequential circuits, there is no existing method which could help in selecting the random vectors that should be conserved, without affecting fault coverage.

In our approach, we use testability measures to estimate the number of time-frames needed to test any fault of the sequential circuit. Based on TMs, it can be shown that, in

most of the cases, it is possible to conserve only the sequences that detect faults without affecting the fault coverage.

Mixed (random and deterministic) vector generation in a sequential ATPG, which we are introducing here for the first time, has two main advantages. First, the number of faults to be considered by deterministic generation will be reduced and test vector generation will be accelerated. Second, the compacted pseudo-random vectors can be used as an initialization sequence that sets the circuit in a partially known state. Starting from this state, ROBDD generation and deterministic generation become much easier and much faster. In other words, after a partial reset, the number of vectors needed will, generally, be reduced.

In this paper, a new sequential ATPG tool is presented. This tool uses mixed (random and deterministic) test pattern generation, an iterative array model [50] and ROBDD representation [1]. The random test vector generation and the sequential variable ordering are based on iterative array length estimation using TMs [9][10][11]. The random test vector generation will be used for the detection of easy-to-test faults. This will reduce the number of faults to be considered by the deterministic test vector generator. The deterministic generation is performed for the remaining faults using function manipulations that are based on ROBDD generation.

The paper is organized as follows: The next section deals with sequential TMs and their use in random pattern compaction. Section 6.3 gives an overview of ROBDD generation for combinational circuits and an extension to sequential circuits is proposed. A new technique for sequential ROBDD generation is proposed. The test generation technique and some experimental results are given in sections 6.4 and 6.5, respectively.

6.1.1 Sequential Circuit modeling

In this paper, we show how our test generation method for combinational circuits can be extended to sequential circuits. Here, we restrict ourselves to those synchronous sequential circuits whose components are gates and clocked flip-flops (F/Fs). The method is based on the modeling technique that transforms a synchronous sequential circuit into an iterative combinational array (Figure 6.1). One cell of each array is called a *time-frame*. In this transformation, a flip flop is modeled as a combinational element having an additional input y to represent its current state and an additional output y^+ to represent its next state, which becomes the current state in the next *time-frame*. An input vector of the combinational array represents an input sequence for the sequential circuit.

Because all the cells in the iterative array have an identical structure, there is no need to actually construct the complete model of the iterative array. Thus our algorithm can use the same structural model for every *time-frame*. Therefore, for BDD generation or for testability measure computations, a synchronous sequential circuit S can be modeled by a pseudo-combinational iterative array, as illustrated in Figure 6.1. In this model, each cell $C(i)$ of the array is a combinational circuit. If an input sequence $x(0), x(1),....x(k)$ is applied to S, which is in an initial state $y(0)$, S generates the output sequence $z(0),z(1)...z(k)$ as well as states $y(1), y(2)....y(k)$ (for sequential nodes). The iterative array will generate the output $z(i)$ from cell i, in response to the input $x(i)$ to cell i ($1 \leq i \leq k$). Note that the first cell also receives the values corresponding to $y(0)$ as inputs. In this transformation, the clocked F/Fs are modeled as combinational elements, referred to as pseudo-F/Fs. The present state y of the F/Fs in cell i must be equal to the y^+ output of the F/Fs in cell i-1. This modeling technique maps the time domain response of the sequential circuit into a space domain response of the iterative array.

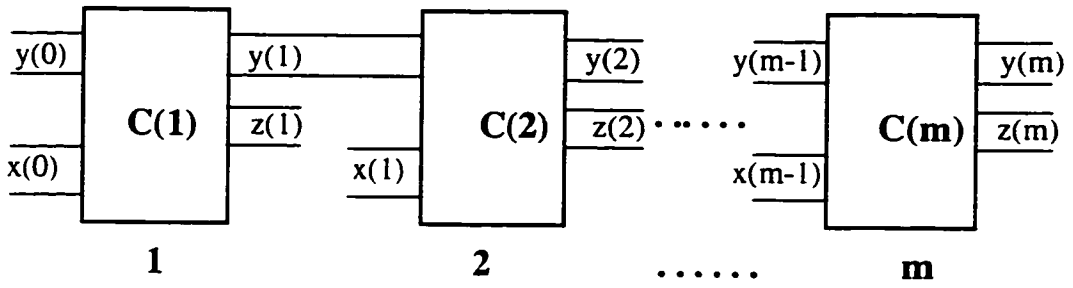


Figure 6.1 Iterative array model of a sequential circuit

An important question that often arises is when to stop increasing the number of time-frames.

For a circuit with n F/Fs, for every one of the 2^n fault-free states, the faulty circuit may be in one of its 2^n states. Hence no test sequence without repeated states can be longer than 4^n [52]. This bound is too large to be useful in practice. So, the number of time-frames must be bounded by a (much tighter) user-specified limit f_{max} or a limit computed by using testability measures.

6.2 Sequential TMs and random vector generation

Probabilistic TMs have been efficiently used in combinational circuit testability analysis and fault coverage estimation. Sequential circuit testing is different from combinational circuit testing. In fact, at the beginning of the test vector generation process, sequential nodes are considered to be in unknown states. To generate a test vector to a given fault, some of the sequential nodes have to be in known states. In order to estimate the difficulty of bringing a sequential node to a known value in a pseudo-random context, given that it starts from an unknown value, sequential controllability, called initializability, has been introduced in [9] and [10]. Using sequential controllability, observability and

the iterative array model of Figure 6.1, the fault coverage of a sequential circuit can be estimated. Using these measures, the maximum length of the iterative array can be easily estimated. In fact, we only need to find the number of duplications after which the controllabilities, or the estimated fault coverage, stop increasing

In [10], it has been shown that sequential circuits can be tested by pseudo-random vectors. It has also been shown that by using an estimation for the iterative array length, a compacted set, that allows the same fault coverage, can be found in most of the cases.

6.2.1 Sequential TMs

In this section, the probabilistic TMs for sequential circuits and their use in pseudo-random vector compaction are reviewed. Initializability, originally developed for functional modules [9], can as well be used for sequential circuits testability evaluation [10] and [11].

Initializability is a quantification of the difficulty in controlling a sequential node given that it starts from an unknown state. With the initializability concept, TMs can be computed for combinational and sequential modules. The initializability measure is a probabilistic measure whose computation is similar to COP number computation [8]. Using an iterative array model, the sequential controllabilities are computed as a function of time.

Soufi et al. [11] have used these measures for partial set and reset in order to enhance the testability of the circuit and to make it fully testable, using only pseudo-random test vectors. Based on Markov chain theory, it has been shown that these measures are non-decreasing functions of time.

We can distinguish three different controllabilities, denoted by $C_0(h, t)$, $C_1(h, t)$ and, $C_X(h, t)$ which represent the probabilities that h take the logic value 0, 1 or X (unknown), after applying t pseudo-random test vectors respectively. Since the sum of these three terms is always equal to 1, only two of them are to be computed; the third can be deduced from the others.

Here, only C_0 and C_1 will be computed [11]. Initially (at time 0), the circuit is supposed to be in an unknown state. In this case, the controllabilities C_0 and C_1 of all the flip-flops are set to 0 (which means that $C_X = 1$). However, since the primary inputs can be accessed at any time t and the vector applied is randomly chosen, then their values can be set to 0 or to 1 with an equal probability. Hence, C_0 and C_1 of a primary input are both set to 0.5.

For the other lines of the circuit, $C_0(h, t)$ and $C_1(h, t)$ are computed using the formulas given in [11]. These formulas depend on the type of gate. However, for a flip-flop F, and at any time t ($t > 0$), $C_0(F, t) = C_0(I, t-1)$ and $C_1(F, t) = C_1(I, t-1)$, where I is the input of the flip-flop and F is its output. In other words, the controllabilities computed for the input line of a flip-flop at time t will be the controllabilities of the flip-flop output at time $t+1$. These measures can then be used as a stopping criterion in the iterative model. The formulas for the computation of sequential TMs are presented in [11]. These TMs are used by our algorithm, first, for the ordering of the input variables and second for selecting the random vectors that have to be conserved.

6.2.2 Random vector generation and compaction

For sequential circuits, the pseudo-random vector generation step has not been used up to now, mainly because the number of pseudo-random vectors is huge and no method

has been developed so far to reduce this number. For combinational circuits, pseudo-random vector compaction is straight-forward: only those vectors that detect faults are retained. On the other hand, for sequential circuits, to test a fault, a sequence of test vectors is generally needed. Since the length of the sequence is generally different for each fault, the pseudo-random vectors cannot be compacted efficiently. In other words, when the compaction is performed regardless of the maximum sequence length, the fault coverages obtained before and after compaction will be different.

Our compaction technique is based on the fact that maximum length of the test vector sequences can be estimated using TMs. It can be shown that, using such measures, and in most of the cases, the pseudo-random vectors can be easily compacted without affecting the fault coverage.

In order to estimate the maximum length of test vector sequences, we have to find a stopping criterion based on testability measures. It has been proven in [11] that the probabilities C_0 and C_1 of any node are non-decreasing functions of time. In our case, a controllability is considered stable whenever the difference between this controllability at time t and that at time t-1 is less than ε (In this paper, ε is equal to 10^{-4}). We have found that choosing ε less than 10^{-4} does not improve our results. Suppose that, at time t, the controllabilities computed for a gate of a sequential circuit remain stable (the difference is less than ε). This indicates that by applying any sequence of $t+1$ vectors, v_1 to v_{t+1}, the vector v_1 will have no effect (with a probability close to 1) on the logic value taken by the flip-flops (or the state of the sequential circuit). This also means that even if v_1 is removed from this sequence, the state of the circuit will be the same with a high probability.

The maximum length of a test vector sequence is the number of circuit duplications found by TMs evaluation. In our case, the maximum length of a test vector sequence is the number of duplications for which the controllabilities reach a stable values.

Here, we will only give an overview of the method presented in [10]. The method is based on fault simulation. Before going further, let us suppose that, using the TMs, we find that the maximum length (which represents the maximum number of time-frames to be considered) is n.

Suppose that the fault simulator declares a fault detected after applying a sequence of pseudo-random vectors, and for the last M vectors, the fault simulator didn't declare any other fault. Then, two different cases have to be considered:

Case 1: When $M <= n$, all the M vectors are useful for the detection of the fault. Consequently, all the M vectors must be conserved.

Case 2: When $M > n$, according to the TMs, only the last n vectors have to be conserved, since the maximum length is n. In this way, the set of vectors will be compact and will often allow the same fault coverage. In addition, using a parallel fault simulator, like *hope*, the time spent in simulation will be reduced, and pseudo-random vector generation can be used by sequential ATPGs, as a first step, for the detection of easy-to-test faults. The compacted set thus obtained, can be also used as an initialization sequence for deterministic test vector generation.

Our compaction procedure is applied for some ISCAS'89 benchmark circuits. Table 6.1 shows the fault coverage given by the parallel fault simulator *Hope* [52] before and after compaction (FC-B and FC-A) for each of the circuits that has been studied. It

shows also the number of vectors after compaction (NBV) and the reduction (R), in percentage, of pseudo-random vectors.

Table 6.1 Pseudo-random vector compaction

Circuit	FC-B(%)	FC-A(%)	NBV	R(%)
s208	43.256	43.256	150	99.925
s298	85.39	85.39	395	99.8
s344	96.199	96.199	91	99.915
s382	13.534	13.534	51	99.974
s386	73.438	73.438	342	99.829
s420	31.68	31.68	203	99.898
s444	15.401	15.401	69	99.965
s526	11.351	11.351	52	99.974
s641	86.081	86.081	343	99.828
s820	49.647	49.647	649	99.675
s838	23.804	23.804	649	99.675
s1488	77.658	82.234	554	99.723
s1494	79.017	79.017	984	99.508
s5378	61.786	61.786	530	99.735
s9234	11.199	11.199	42	99.979
s13207	9.129	9.129	187	99.906
s15850	33.785	33.785	3245	98.377
s35932	86.077	86.701	960	99.52

Table 6.1 shows that the FC can be easily reproduced. In this table, we have presented for each circuit, the fault coverage obtained after applying 200000 pseudo-random vectors. According to the same table we note that, for all the considered circuits, the fault coverages before and after compaction are exactly the same and the number of vectors is reduced dramatically, with a 99.739% reduction on the average.

These results show clearly that sequential circuit testability can be easily estimated using TMs. In addition, pseudo-random vector compaction can be easily performed [10] and implemented. We need only to conserve the sequences of vectors that detect faults.

6.3 Sequential ROBDD Generation

ROBDD representation has been used in many applications. Among them we can mention formal verification and test vector generation. However, this kind of representation was limited only to combinational circuits. Because sequential circuit functionality depends on time, time has to be taken into consideration in order to build a BDD that represents the functionality of any node of the sequential circuit.

In order to generate BDDs for sequential circuits, the iterative array model is used. In the first time-frame, we assume that the circuit starts from an unknown state. In other words, the logic values of the flip flops are X (unknown). The primary inputs of the sequential circuit can be set to 0 or to 1 at each time frame. In this case, given an assignment to the sequential primary inputs, a node of the circuit can take the logic value 0, 1 or X. The BDD must also represent these three different logic values. Consequently, to represent the BDD of any node of a sequential circuit after t time frames, three terminal nodes (0, 1, and X) are needed. In addition, we must have the information about time in each node of the BDD, since the logic values of the same sequential primary input are time independent.

The BDD sizes depend greatly on variable ordering. A good variable ordering can be found using the TMs measured as discussed in Section 6.2.1. In addition, BDD sizes depend also on the complexity of the circuit being considered. Using an iterative array model, the complexity of the circuit considered increases with the number of duplications, since the number of primary inputs (nodes) will be equal to the number of primary inputs (nodes) of the sequential circuit multiplied by the number of time-frames (duplications).

In order to have manageable BDDs and to generate test vectors for sequential circuits, the number of duplications has to be limited. The TMs will be also used for this purpose. In addition, the use of the compacted pseudo-random vectors, for initializing the sequential circuit, will reduce the complexity of the ROBDDs, since the number of duplications needed for finding a test vector will generally be reduced. In other words, in most of the cases, fewer vectors will be needed when we start from a known rather from a completely unknown state.

6.3.1 Review of Combinational ROBDD

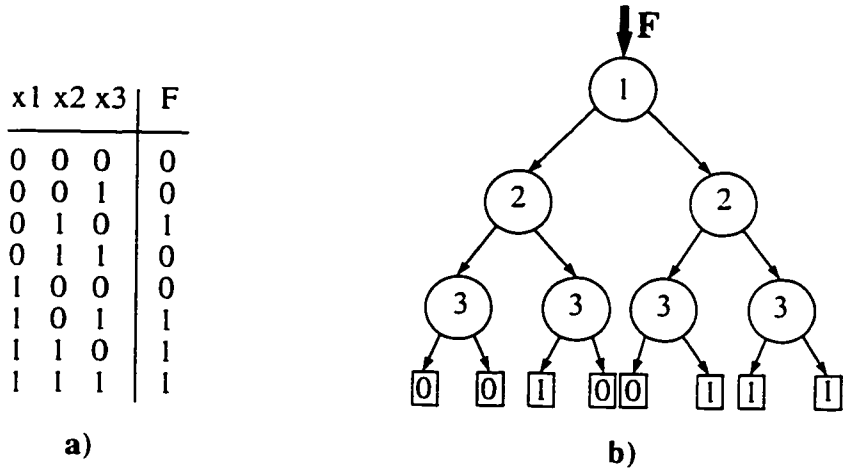
The fundamental principles of ROBDD generation for combinational circuits are reviewed in this section.

Given a boolean function F and an ordering of decision variables $(x_1, x_2, \ldots, x_i, \ldots x_n)$, an OBDD is the graph representation of the Shannon expansion of F according to the given variable ordering. For example, the OBDD of the function $F(x_1, x_2, x_3)$ described in the truth table in Figure 6.2a is shown in Figure 6.2b. In this figure, each non-terminal vertex is represented by a circle containing the index of a decision variable and each terminal vertex is represented by a square containing the function value 0 or 1. Although, the "fully expanded" OBDD requires $2^n - 1$ non-terminal vertices, its graph size can be drastically reduced by using the following two reduction rules:

1) Merging rule: two isomorphic subgraphs should be merged.

2) Deletion rule: a vertex whose two branches point to the same vertex should be deleted.

Bryant [1] repeatedly applied these two reduction rules to an OBDD until both of them were no longer applicable. The resultant graph is called the reduced OBDD (ROBDD) and is unique [1]. For instance, Figure 6.3 is the ROBDD of Figure 6.2b.



x1	x2	x3	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 6.2 A Boolean function and its OBDD

Given a boolean function G, the ROBDD nodes for G and $\overline{G}$ are similar except that their terminal nodes are interchanged. This similarity can be exploited by using complement edges. Therefore, $\overline{G}$ could be represented by a complement edge to the node for G, saving intermediate nodes. In addition to reducing the BDD sizes, complement edges make boolean function manipulation easier.

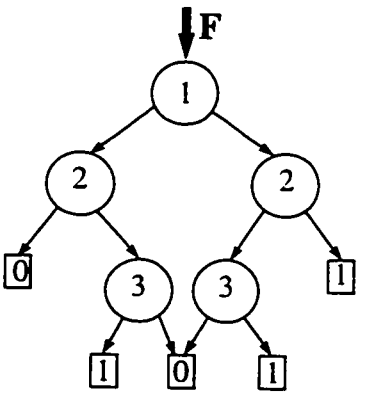


Figure 6.3 ROBDD with only regular edges

To maintain a canonical form, we must have some control on where complement edges are placed. Karpulus [4], Mardre [5] and Karl [6] formulated a set of rules to guarantee canonical ROBDDs using complement edges. Our implementation uses the rule presented in [6]: The high edge of every node must be a regular edge. Thus, we always choose the left member of each pair of functions, as shown below (Figure 6.4). Note that these 4 pairs are functionally equivalent. A dot on an edge is used to indicate that it is a complement edge.

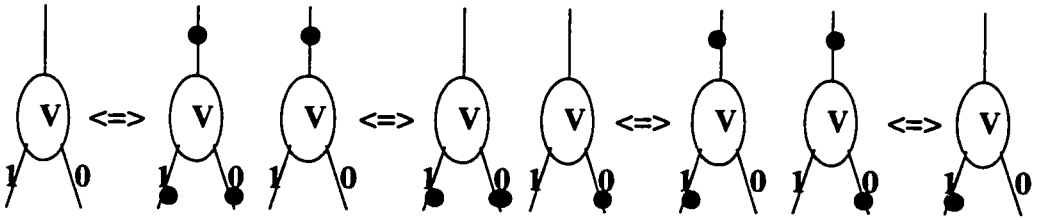


Figure 6.4 Equivalent functions

Using complement edges with the constraint described above, the size (number of nodes) of the graph in Figure 6.3 can be reduced. The ROBDD graph of Figure 6.5 repre-

sents the same boolean function F illustrated in Figure 6.2, with 4 nodes fewer than in the graphs shown in Figure 6.2b.

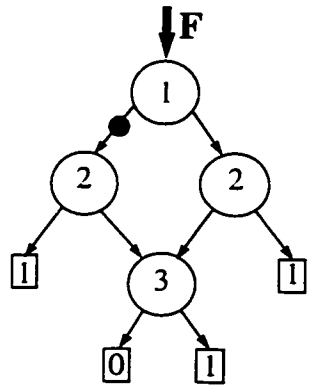


Figure 6.5 Reduced OBDD with the two kinds of edges

6.3.2 Sequential OBDD construction

The use of the iterative array model makes it possible to use the same BDD generation strategy developed for combinational circuits. The extension to synchronous sequential circuits is made with the following two modifications:

1. Function manipulation is performed with the unknown states X.

2. Time is included in BDD nodes.

Since the circuit *C'*, obtained by adding the F/F models to the original circuit *C*, is also a combinational circuit, our BDD generation method can be adapted in order to take into consideration the unknown state. We also have to take into account the fact that the same variable can take different logic values at different times, as if we are dealing with

two different inputs. The BDD generated is also bounded by the maximum length of test vector sequence estimated using TMs.

6.3.3 Variable ordering

Here, we will see how the testability measures introduced in section 2.1 can be used to find a good ordering for the input variables of a sequential circuit. In order to find a good ordering for the input variables while computing the controllabilities, we register for each gate of the circuit the time t when all its controllabilities become stable. Then, if we are only interested in finding the BDD at this gate, there is no need to iterate more than t times. We also know that, by applying a sequence of $t+1$ vectors, the first vector will have no influence (with a probability close to 1) on the final state of the circuit. Further, in order to have a reduced BDD, the inputs which are close to the primary outputs must be ordered before other inputs. Using this approach, the size of the BDDs generated will be reduced. In other words, according to the iterative array model, an input A at time t (or $A[t]$) must be ordered before the same input at time t-1 ($A[t-1]$).

Now, we have to present a strategy for ordering the variables of the same time-frame. In this paper, we mean by sequential depth of a net the time when all its controllabilities become stable. To order the inputs of a sequential circuit, we first order the inputs of each gate according to their sequential depths from the lowest level to the highest. In this way, the net with the lowest sequential depth is traversed first while ordering the variables. Second, we traverse the circuit from the output considered to the inputs in a depth first search manner. The first input reached will be made first in the BDD ordering, the second input reached will be ordered second, and so on until all the inputs that can be reached from this output are ordered.

Example 1 deals with ROBDD generation when the circuit starts from an unknown state. The variable ordering is performed according to the time-frames and the sequential depths of circuit nodes.

Example 1

The circuit of Figure 6.6a is used to illustrate the ROBDD generation for a sequential circuit starting from an unknown state X. It also illustrates the effect of the ordering on BDD size. The ROBDDs of the output *Po* are generated in time-frames 0, 1 and 2, and are presented in Figure 6.6b, 6.6c and 6.6d, respectively. The variable *A[0]* represents the input *A* at time-frame 0.

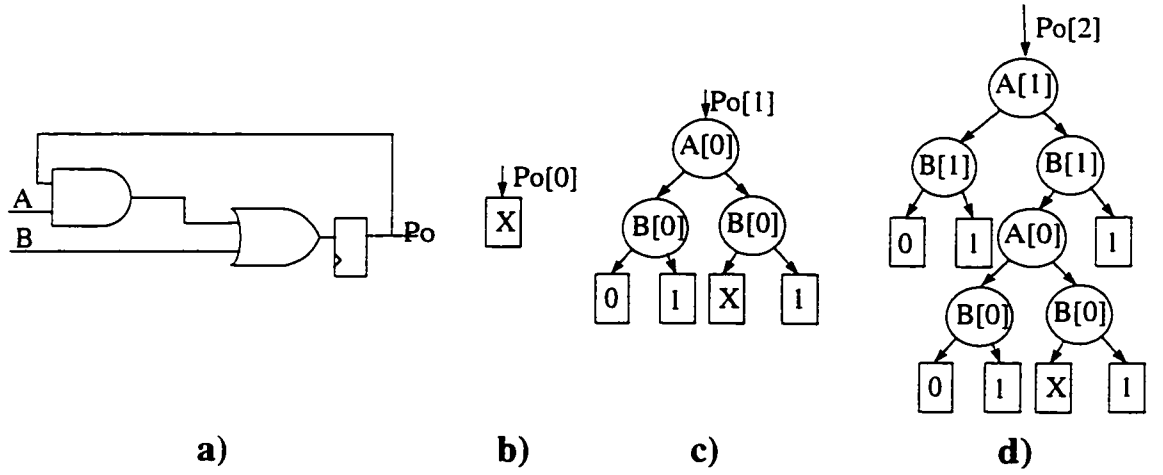


Figure 6.6 ROBDD of Po in time-frame 0, 1 and 2

In *time-frame* 0, the primary output *Po*, which is also the output of the flip-flop, starts from an unknown state (Figure 6.7b). In time-frames 1 and 2, Po can be set to one or to zero since there is a path which leads to zero and a path which leads to one in the BDD in

Figures 6.6c and 6.6d. Figure 6.6d the variables in time-frame 1 (*A[1] and B[1]*) are ordered first, followed by the variables in time 0 (i.e. A[0] and B[0]). When the sequential depth of each net is considered in variable ordering, BDD sizes can be reduced. For the circuit of Figure 6.6a, the sequential depth *B* is less than the sequential depth of *node C*. In Figures 6.6c and 6.6d, the input variable A is ordered before the input variable *B* in each time-frame. In Figures 6.7a and 6.7b, we show the case when the sequential depth is taken into consideration. By ordering *B* before *A*, as illustrated in Figure 6.7, the BDD size is reduced.

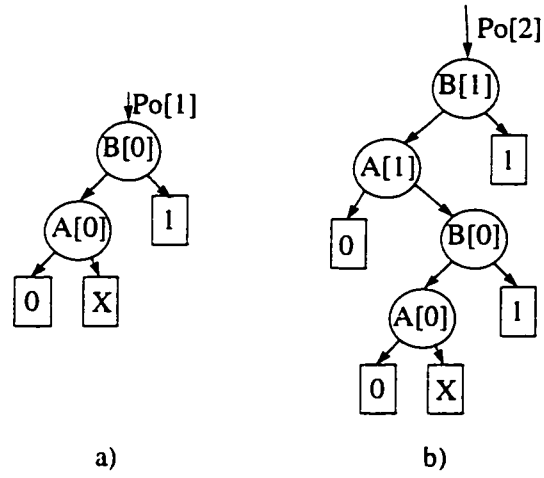


Figure 6.7 ROBDD of Po in time-frame 1 and 2 when the sequential depth is taken into consideration

Example 2

In this example, we study the case of ROBDD generation when the circuit starts from a known state. The *s27* circuit (Figure 6.8a) is used to show the sequential ROBDD construction after applying a set of vectors. A set V={1011,0100,0100,1010} of five random

vectors is applied to the s27 circuit, 1011 is applied first, 0100 second, and so on. Then, the sequential nodes (15,16,17) of the fault-free circuit will be at state (1,0,0) after applying V. After two time-frames, the BDD of the output Po will be as shown in Figure 6.8b. Note that the BDD does not contain any terminal node X, since the circuit was set in a known state by the test sequence V. The variable C[0] represents the input C at time-frame 0, just after initializing the circuit. Thus, the generated BDD contains the time information and can also contain the unknown terminal node.

According to Figure 6.8b, the ROBDD generated at time-frame 1 contains the inputs applied at time-frame 0, which is a well known characteristic of synchronous sequential circuits.

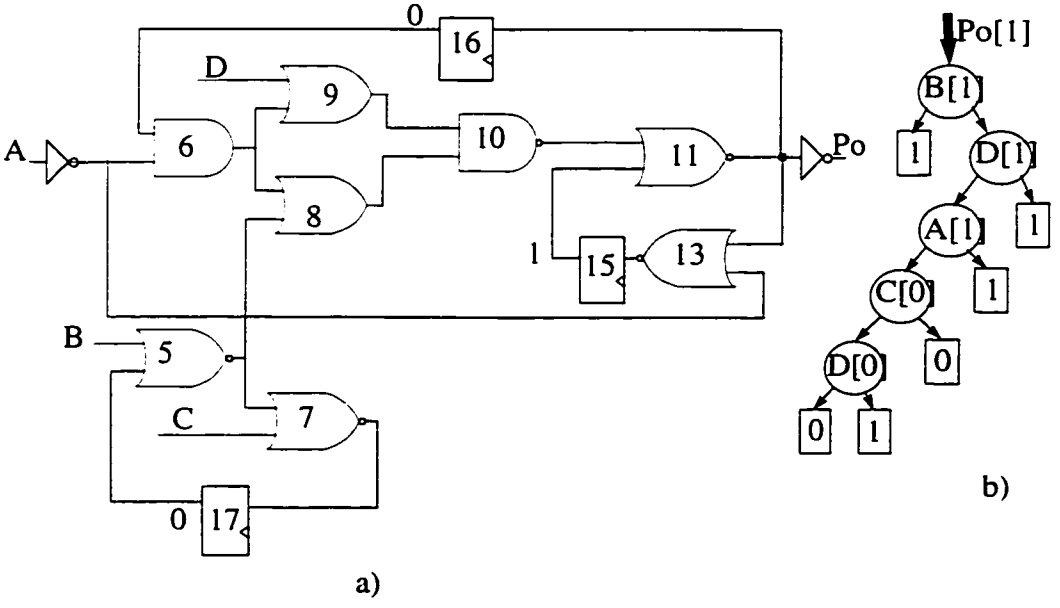


Figure 6.8 s27 circuit and its ROBDD in time-frame 1

6.4 Test vector generation for sequential circuits

To test a fault, our algorithm finds the corresponding test set, the set of inputs for which the output of the faulty circuit is different from that of the fault-free circuit. If F_α is the function implemented by the faulty circuit and F is the function implemented by the fault-free circuit, then the set of inputs for which F_α is not equal to F is found by satisfying the formula $F \oplus F_\alpha = 1$. Then, when ($F \oplus F_\alpha = 0$ or $F \oplus F_\alpha = X$) after n time-frames, no sequence of n test vectors that tests the fault.

Since generally it is not possible to find a single good ordering for all the primary inputs of a circuit, in our approach, the primary outputs (POs) of the sequential circuit are studied separately. The main advantage of this approach is to study larger circuits. The only disadvantage is that the CPU time used be greater than that corresponding to the case where a single good ordering can be found easily for small circuits. Since the BDDs will be generated for many time frames, the number of the variables that must be considered will increase. Consequently, it is very convenient to consider the primary outputs separately.

In our approach, a number of pseudo-random test vectors is first applied to the sequential circuit to initialize the circuit and generate test vectors to the easily tested faults. The state reached (the logic values taken by the flip-flops) of the fault-free circuit is taken into consideration in BDDs' construction. Then, when the value taken by a flip flop is 0, in the first time frame, terminal node 0 is returned whenever this flip-flop is reached while constructing the BDDs of the fault-free circuit. Generally, a different state is reached for each faulty circuit. Next, for each fault, we have to compute the faulty state

reached after applying the same set of test vectors These states must also be taken into consideration while generating the BDDs for faulty circuits.

After selecting a primary output (PO), the input variables of the sequential circuit are ordered according to the sequential depths of the circuit nodes. We take a fault F and check if its corresponding line can be reached by the considered PO. If this is the case, we first compute the fault-free and the faulty BDDs of the primary output at time frame 0. If the BDDs computed at time frame 0 are equal, then it is impossible to test the fault by adding only one vector to the set. In this case, we compute the fault-free and faulty BDDs at time frame 1. When the XOR of the functions corresponding to the faulty and fault-free BDDs of the output can be made equal to 1, a set of test vectors can be found. The BDD of the output of a flip-flop at time frame t is made equal to the BDD computed for its input at time frame t-1.

A test vector is obtained by making the boolean function of the faulty circuit different from that of the fault-free circuit. To generate a test vector, we traverse the two BDDs corresponding to these boolean functions with the same variable assignment until the terminal nodes reached by both BDDs are different. While traversing the BDDs, we compare the BDDs that will be referred depending on the assignment. Then, if the logic value 1 is assigned at time t to the i^{th} primary input, then we always follow the edge 1 (high) of any vertices whose index is i and the corresponding time is t, while traversing the BDDs. If, for example, with $x_i = 1$, the BDDs referred are different, then we continue the traversing. In the other case, if no test vector can be produced using such an assignment, then, we must change the variable assignment. We set x_i to 0 and repeat the experiment. If with this assignment, the referred BDDs are also equal, then we change the assignment of the vari-

able x_{i-1}. We proceed in this way until we reach different terminal nodes. Using this method, a test vector is produced in a linear time. The test vector generation ends when all POs have been studied. The undetected faults cannot be detected by any combination of n test vectors, where n is the number of time frames considered.

Example 3

Suppose that we have to test fault G_{s-a-1} (line G stuck at 1) of the circuit shown in Figure 6.9. Suppose also that the same sequence V of test vectors, presented in example 2, is applied to this faulty circuit.

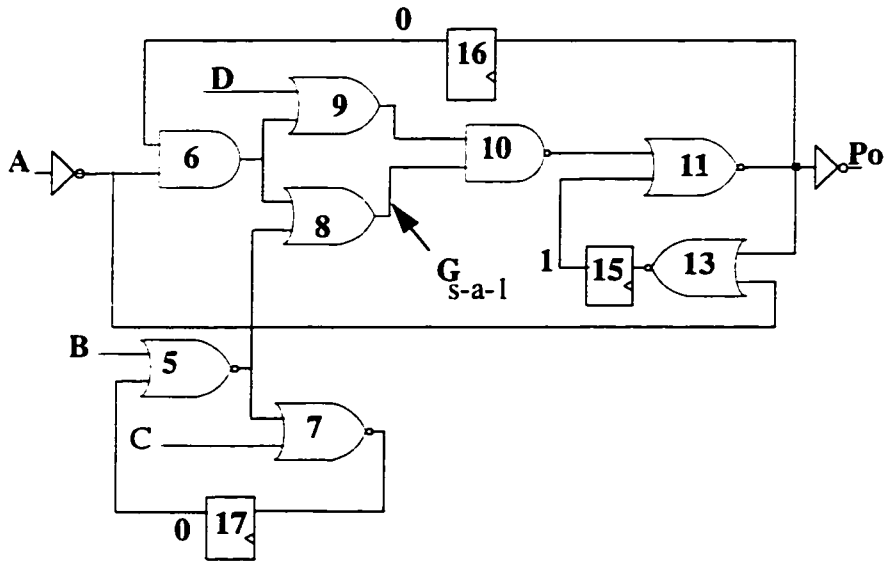


Figure 6.9 s27 circuit with fault G_{s-a-1}.

To find a test vector for the fault G_{s-a-1}, we compute the boolean functions corresponding respectively to the fault-free circuit and to the faulty one (with fault G_{s-a-1}), at different time-frames.

In this example, we have found that the state reached, after the initialization sequence, is the same for both the faulty and the fault-free circuits. The BDDs of the faulty circuit at time-frame 1 are given in Figure 6.10. In time-frame 0, the faulty and fault-free BDDs are equal, but in time-frame 1 they are different (Figures 6.8b and 6.10).

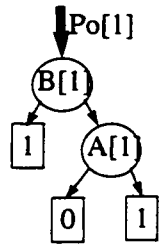


Figure 6.10 ROBDD of s27 circuit with fault G_{s-a-1} in time-frame 1

In order to find the test vector that detects G_{s-a-1}, the XOR of the faulty and faulty-free BDDs after two frames are constructed (Figure 6.11).

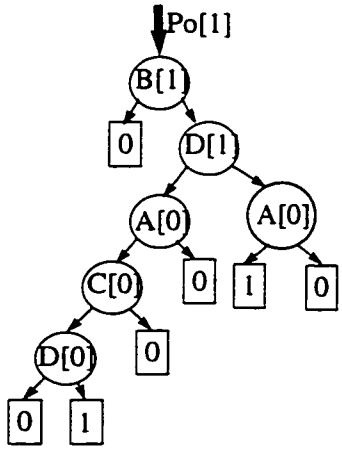


Figure 6.11 ROBDD of the Xor of the faulty and fault-free circuits.

According to Figure 6.11, one of the sequences of vectors which tests G_{s-a-1} is {0XXX,X1X1} since it leads to 1. Note that this sequence must be applied after the set V in order to test the fault G_{s-a-1}. Then the set V2 = {1011, 0100, 0100, 1010, 0XXX, X1X1} allows to test this fault irrespective of the initial state of the circuit.

6.5 Experimental results

In order to show the effectiveness of our test generation technique, TMs, test vector compaction, sequential ROBDD generation and test vector generation are implemented. Our ATPG, whose flow-chart is given in Figure 6.12, is composed of four major steps:

1) Testability measure computation and iterative array length estimation, 2) Random vector generation and compaction, 3) BDD generation and ordering, and 4) Deterministic test vector generation and fault simulation.

After fault-collapsing, two phases of test pattern generation follow: random and algorithmic. The first phase of test pattern generation is the random phase. Here we use the fault simulator *hope* [52]. In this way, we generate patterns for the easy-to-test faults. When *hope* completes the second phase, algorithmic test pattern generation begins. During this algorithmic phase, each pattern generated is simulated. In this way, any fault accidentally detected by the new pattern is removed from the fault list.

Our ATPG algorithm has been implemented in C language and has been run on ISCAS'89 [12] sequential benchmark circuits. In Table 6.2, CPU time, fault coverage and number of vectors, obtained by running our algorithm, are compared with the corresponding results obtained three existing methods. The results for these three existing methods are taken from [48].

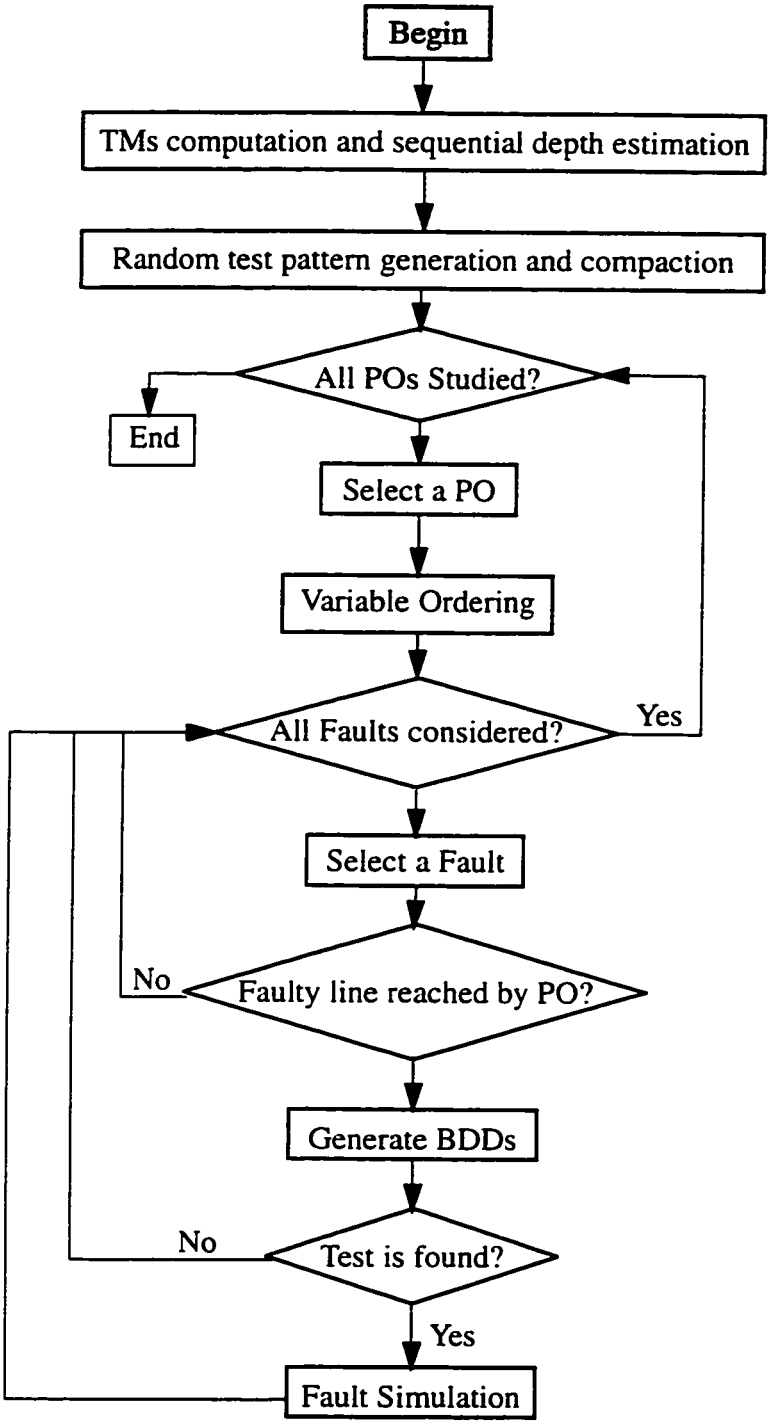


Figure 6.12 Flow-chart of the test vector generation algorithm

Table 6.2 Test generation result comparison

	CPU time				Test length				Fault coverage			
Circuit	HITEC	LEE	MER.	st_bdd	HITEC	LEE	MER.	st_BDD	HITEC	LEE	MER.	st_BDD
s208	29	49	16	135	184	194	160	221	63.7	63.7	63.7	63.7
s298	15969	3250	1570	8165	306	203	207	350	86	85.7	86	86
s344	4787	354	34	217	142	64	78	91	95.9	96.1	96.1	96.2
s349	3235	407	27	198	137	84	83	150	95.7	95.7	95.7	95.7
s382	43197	3278	538	4294	4939	1286	1109	1213	90.9	91.2	91.2	90
s386	82	82	23	128	311	292	232	350	81.7	81.7	81.7	81.7
s400	43587	3817	515	3295	4309	1098	1001	1438	89.9	90.2	90.2	90.2
s420	17015	1034	207	2543	120	172	167	156	41.3	41.6	41.6	41.6
s444	58131	3252	1276	5432	2240	840	897	873	87.3	89	89	89
s526	168131	13434	7651	14673	2232	1919	1900	1877	65.7	81.2	81.2	78.5
s641	1083	28	9	176	216	185	174	341	86.5	86.3	86.3	86.9
s713	95	242	39	245	194	175	149	234	81.9	81.7	81.7	81.7
s820	5806	4096	882	6472	984	903	902	1432	95.6	95.5	95.8	95.5
s832	6350	3911	1303	8488	981	905	905	1364	93.6	94	94	93.6
s1196	102	149	44	57	453	376	352	348	99.7	99.7	99.7	99.7
s1238	144	195	59	149	478	372	362	355	94.6	94.6	94.6	94.6
s1488	13348	2749	529	9382	1294	1270	979	1347	97.1	97.1	97.1	97.1
s1494	6981	2345	368	2591	1407	1163	850	1587	96.4	96.4	96.4	96.4

According to Table 6.2, we note that, using our approach, we reach about the same fault coverage than the other approaches. We note also that, in most of the cases, the sizes of test vector sets are as greater as those obtained using the other approaches. Note that, using our approach, the length of test vector sequences can be reduced more. Since, in many cases during the compaction of random vectors, more vectors than needed are conserved. We have only to reduce the number of random vectors applied.

Table 6.2 shows that our algorithm uses more CPU time than that presented in [48]. This is due to the fact that the BDDs to manipulate increase in each time frame, since the

number of variables of the iterative array increases with the number of time frames. In addition, when a new test vector is found, a new simulation is needed to know the new states corresponding to the fault-free and faulty circuits. Consequently, all the BDDs must be re-computed based on the new state of each circuit.

Note that BDD construction time is included in the CPU time shown in Table 6.2. Thus, it is normal that our algorithm uses more CPU time than that presented in [48]. It is important to stress that the BDDs generated can be used in other applications, like formal verifications. In this way, the BDDs are generated only once and can be used for many applications. We also have to mention that, since the primary outputs are considered separately, an undetected fault can often be studied many times by all the primary outputs that reach the faulty line.

The main advantage of our approach is that we can determine the minimum length of the sequence of vectors that must be applied to a sequential circuit to test a given fault, whatever the states of the faulty and fault-free circuits are. For example, for the circuit s208, we have proven that it is impossible to test each of the remaining untested faults with a sequence of less than 25 vectors.

Note also that, we have proven that, only by using our approach of function manipulations, no fault of the s510 benchmark circuit can be tested with any sequence of 100 vectors or less. This information cannot be obtained easily using the other approaches.

6.6 Conclusion

In this paper we have shown that it is possible to have an efficient approach that regroups random and deterministic test vector generations. A compaction technique was

used for the pseudo-random vectors generated, in order to conserve the sequences that detect faults. We have also shown that the compacted set of vectors can be used as an initialization sequence to accelerate the deterministic test vector generator. This reduced the sizes of the ROBDDs and the number of faults considered in the deterministic vector generation.

Our algebraic approach is based on the construction of the of the ROBDDs of the faulty and fault-free circuits. Using this approach, the undetectable faults are studied as easily as the detectable faults. We can determine also the minimum length of the sequence of vectors that must be applied to a sequential circuit to test a given fault, whatever the states of the faulty and fault-free circuits are.

We should also mention that our method can be expanded in many directions. For instance, we can use the dominator concept to avoid studying a fault more than once. Also the number of vectors can be reduced by using only deterministic generation (without using random generation) since in many cases more vectors than needed will be conserved. In fact, after random generation and compaction, our algorithm starts with a relatively large number of test vectors. Thus according to the needs, a trade-off between random and deterministic generation can be investigated.

CHAPITRE 7

CONCLUSION

Dans le chapitre 2, nous avons présenté un ensemble d'algorithmes simples et efficaces qui peuvent faire une variété d'opérations sur les fonctions logiques. En combinant des concepts de l'algèbre booléenne avec des techniques de la théorie des graphes, nous avons obtenu un haut degré d'efficacité. Notre approche utilise des arcs complémentaires pour réduire les tailles des ROBDD et accélérer nos algorithmes.

Nous avons présenté les modifications qui doivent être faites pour étudier les circuits séquentiels. Nous avons démontré qu'en utilisant un modèle itératif pour modéliser le circuit séquentiel, des BDD peuvent être générés pour ces types de circuits.

Nous avons remarqué qu'en évaluant seulement le BDD d'un noeud quelconque, nous pouvons dire quel est le nombre minimum, s'il en existe un, de vecteurs à appliquer pour mettre ce noeud donné dans un état connu. Cette information est très utile dans la génération de vecteurs de test. Le seul inconvénient de cette approche est qu'elle ne peut pas être applicable à des circuits de plus grandes complexités, puisque, généralement, une longue séquence de vecteurs sera requise pour tester une panne donnée.

Dans le chapitre 3, nous avons présenté une technique algébrique pour générer des vecteurs de test pour les circuits combinatoires. Elle est basée sur le modèle de pannes de type collée-à. Elle suppose qu'il existe au plus une panne dans un circuit.

Nous avons démontré qu'avec notre approche les pannes redondantes sont détectées rapidement et des vecteurs de test peuvent être générés pour les pannes difficilement testables aussi simplement que les pannes dites facilement-testables.

Les résultats expérimentaux réalisés sur des circuits d'essai montrent très bien l'efficacité de notre approche par rapport aux autres. Notre approche présente beaucoup d'avantages par rapport aux autres méthodes algébriques. Elle utilise la notion de dominance(dominators) pour éliminer les retour arrières et accélérer la génération de vecteurs de test.

De plus, étant donné le BDD du circuit défectueux et celui du circuit normal (sans présence de pannes), un vecteur de test peut être généré en un temps linéaire. Par conséquent, l'existence d'un vecteur de test peut être déterminée facilement. Nous avons montré comment, en utilisant un ordonnancement variable pour chaque sortie primaire, nous pouvons générer des vecteurs pour des circuits plus complexes.

Dans le chapitre 4, nous avons présenté une technique algébrique de génération de vecteurs de test pour les circuits combinatoires en se basant sur le modèle de pannes collée-à. Le concept de dominance est utilisé pour décomposer le circuit. En d'autres termes, chercher l'ensemble maximum de *supergates*. Ceci a rendu possible la génération hiérarchique de vecteurs de test et a accéléré la génération de vecteurs de test. Notons aussi qu'en décomposant le circuit les tailles des BDD à manipuler sont réduites.

En utilisant des ordonnancements variables pour les sorties primaires, nous avons généré des vecteurs de test pour plus de circuits que les méthodes existantes basées sur la

représentation BDD. En plus, nous avons étudié des circuits plus complexes qui ne peuvent pas être testés en utilisant un ordonnancement unique à toutes les sorties primaires.

Nous avons montré que le concept d'observabilité est très utile dans la génération de vecteurs de test. Nous avons montré que si l'observabilité d'une porte logique, qui est une fonction booléenne, est connue et qu'une des pannes correspondant à une ligne de cette porte a été étudiée, toutes les autres pannes non détectées peuvent être étudiées avec presqu'aucun effort additionnel. Pour tous les circuits analysés, nous avons trouvé notre algorithme plus rapide que les autres approches algébriques et les pannes redondantes sont détectées très rapidement. Il suffit de comparer les BDD du circuit normal et du circuit défectueux. Nous avons trouvé que les pannes difficilement testables sont aussi faciles à tester que les pannes facilement testables.

Les résultats expérimentaux réalisés sur les circuits combinatoires d'essai ISCAS'85 et les circuits séquentiels d'essai ISCAS'89, en supposant que toutes les bascules sont dans une chaîne de balayage (Full Scan), montrent l'efficacité de notre approche.

Dans le Chapitre 5, nous avons démontré que la génération automatique de vecteurs de test est possible pour les circuits mixtes, sans apporter de modifications aux circuits. Nous avons vu comment activer une panne dans le circuit analogique et comment propager facilement l'erreur résultante à travers le bloc numérique. Nous avons montré que, juste en manipulant des fonctions logiques, une assignation permettant de propager l'erreur peut être trouvée.

Pour le test du bloc numérique, nous avons utilisé une approche algébrique basée sur le modèle de pannes collée-à. Nous avons vu l'influence de la dépendance introduite sur la

couverture de pannes en connectant les blocs du circuit mixte. Nous avons trouvé qu'il existe des pannes testables, lorsqu'un circuit est considéré seul, qui deviennent non testables lorsque le circuit fait partie d'un circuit mixte.

Nous avons montré comment, en manipulant seulement les fonctions booléennes, on arrive à prendre en considération les contraintes imposées par chacun des blocs du circuit mixte. Nous avons utilisé la méthode, présentée au chapitre 2, pour manipuler efficacement les fonctions logiques. Les résultats de simulation et une validation pratique ont confirmé l'efficacité de la méthode proposée.

Dans le Chapitre 6, nous avons présenté une nouvelle méthode algébrique pour générer des vecteurs de test pour les circuits mixtes. Nous avons démontré que la génération aléatoire de vecteurs de test peut être utilisée comme une première étape de la génération de vecteurs de test. Elle est utilisée pour tester les pannes dites facilement testables. Ainsi, un nombre réduit de pannes sera considérée par la génération déterministe. Ainsi, la génération de vecteurs de test sera accélérée.

Pour plus accélérer la génération de vecteurs de test, les séquences de vecteurs provenant de la génération aléatoire sont appliquées pour initialiser le circuit. De cette façon, le circuit séquentiel est mis, dans la plupart des cas, dans un état partiellement connu, et par conséquent, les tailles des BDDs à manipuler sont réduites.

Notre approche déterministe de génération de vecteurs de test est basée sur la génération de BDD pour le circuit normal et le circuit défectueux. Elle est basée sur le modèle itératif d'un circuit séquentiel. Le nombre maximum de duplications de ce circuit est

estimé par des mesures de testabilité qui ont été utilisées pour ordonner les variables d'entrée du circuit.

Dans le Chapitre 6, nous avons vu comment, en comparant seulement les BDD générés, on peut déterminer s'il existe un vecteur de test ou non. En se basant seulement sur les BDD, nous pouvons déterminer si une ligne donnée peut être mise dans état connu, nous pouvons déterminer aussi avec certitude quel est le nombre minimum de vecteurs à appliquer pour tester une panne donnée.

Notons que la méthode de génération de vecteurs de test, présentée au chapitre 6, peut être améliorée de plusieurs manières. On peut utiliser, par exemple la notion de dominateur, comme au Chapitre 4, pour éviter d'étudier une panne plus qu'une fois, ce qui accélérera l'algorithme. En plus, nous pouvons réduire le nombre de vecteurs de test aléatoires conservés, puisque dans plusieurs cas on conserve plus de vecteurs qu'il faut. En effet, après la génération aléatoire et la compaction, notre algorithme commence par un grand nombre de vecteurs.

Une solution pour résoudre ce problème consiste à prendre la profondeur séquentielle des lignes de circuits en considération lors de la compaction des vecteurs aléatoires. Dans ce cas, le nombre de vecteurs à garder dépend de la profondeur séquentielle de la ligne correspondante à la panne considérée. L'autre solution est de trouver un compromis entre la génération aléatoire et déterministe.

Notre principale contribution a été le développement d'outils rapides et efficaces pour générer des vecteurs de test pour différents types de circuits. Toutes les nouvelles méthodes de génération de vecteurs de test que nous avons développées pour les circuits combinatoires, séquentiels et mixtes sont basées sur le BDD. Ceci nous a permis d'étudier les

pannes redondantes aussi facilement que les autres pannes et de rendre nos algorithmes plus rapides que les autres approches.

L'originalité de cette thèse réside dans la nouvelle méthode efficace et rapide que nous avons développé pour manipuler les fonctions logiques (qui peuvent être partiellement définies), dans la stratégie que nous utilisons dans la génération de vecteurs de test sans retour arrières pour différents types de circuits, ainsi que dans la modélisation que nous utilisons pour générer des tests pour les circuits mixtes.

Nous estimons que ce travail propose des solutions à des problèmes réels et d'ordre pratique dans le domaine du test. D'autre part, il nous a permis de comprendre les différents aspects du test et les problèmes reliés à ce dernier. De plus, ce travail ouvre une fenêtre pour d'autres axes de recherches. Parmi les recherches qui peuvent être réalisées pour élargir l'utilité de ces travaux: une méthode permettant de propager une erreur due à une panne dans un des blocs analogiques d'un circuit mixte à travers les autres blocs tout en prenant en considération le bruit.

Les notions de dominance et d'observabilité peuvent être utilisées pour accélérer la génération de vecteurs de test pour les circuits séquentiels. Ainsi, on peut générer hiérarchiquement des vecteurs de test pour ces types de circuits.

Finalement, les algorithmes que nous avons développés pour la manipulation des fonctions logiques et la génération de vecteurs de test peuvent être réutilisés en adoptant une approche basée sur le branch-and-bound. De cette façon, l'espace de recherche sera réduit et les pannes redondantes seront détectées facilement, ce qui accélérera la génération de vecteurs de test.

Bibliographie

[1] R.E. BRYANT, "Graph-based Algorithms for Boolean Function Manipulation," IEEE Trans. comput., vol. C-35, pp. 677-691, Aug. 1986.

[2] M. FUJITA, H. FUJISAWA, and N. KAWATO, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagram," Proc. ICCAD, 1988, pp. 2-5.

[3] S. MALIK, A.R WANG, R. K. BRAYTON, and A. SANGIOVANNI-VINCENTELLI, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," Proc. ICCAD, 1988, pp. 6-9.

[4] K. KARPLUS, "Representing Boolean Functions with If-then-Else DaGs," Computer Engineering UCSC-CRL-88-28, UC Santa Cruz, Dec. 1988.

[5] J. C. MARDRE and J.-P. BILLON, "Proving Circuit Correctness Using Formal Comparison Between Expected and Extracted Behavior," Proc. 25th Design Automation Conference, June1988, pp. 205-210.

[6] KARL S. BRACE et al. "Efficient Implementation of a BDD Package," 27th ACM/IEEE Design Automation Conference, 1990, pp. 40-45.

[7] F. BRGLEZ and H. FUJIWARA, "A Neural Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," International Symposium on Circuits and Systems, June 1985.

[8] F. BRGLEZ, P. POWNALL and R. HUM "Applications of Testability Analysis: from ATPG to Critical Delay Path Tracing," Proceeding of the International Test Conference, Philadelphie (Penns.), 1984, p. 705-712.

[9] N. BEN-HAMIDA, B. KAMINSKA and Y. SAVARIA, "Initiability: A Measure of Sequential Testability," ISCAS 1993.

[10] N. BEN-HAMIDA, B. KAMINSKA and Y. SAVARIA, "Pseudo-Random Vector Compaction for Sequential Testability," International Symposium on Circuit and System 1994, pp. 4.63-4.66.

[11] M. SOUFI, Y. SAVARIA, F. DARLAY and B. KAMINSKA "A Pseudo-Random Testing Approach for Both Sequential and Combinational Circuits," Proc. of IEEE International Conference on VLSI, 1993.

[12] BRGLEZ, D. BRYAN, and K. KOZMINSKI, "Combinational Profiles of Sequential Benchmark Circuits," ISCAS, 1989, pp. 1929-1934.

[13] TED STANIAN, "TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation," Proceedings of FTCS 1991, pp. 36-43

[14] H. K. LEE and D. S. HA, "An Efficient Forward Fault Simulation Based on the Parallel Pattern Single Fault Propagation," Proc. of International Test Conference, Oct. 1991.

[15] M. RAY MERCER , ROHIT KAPUR and DON E. ROSS, "BDDs Ordering on Simulated Annealing," Design Automation Conference, June 1992.

[16] JANUSZ RAJSKI, "A Method to Calculate Necessary Assignments in Algorithmic Test Pattern Generation," Proceedings International Test Conference, 1990, pp. 25-34.

[17] P. GOALI, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, vol. C-30, No 3, pp. 215-222, March 1981.

[18] H. FUJIWARA and T. SHIMONO, "On the Acceleration of Test Generation Algorithms," IEEE Transactions on Computers, vol. C-32, no. 12, December, 1983, pp. 1137-1144.

[19] M. H. SCHULZ and E. AUTH, "Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification," IEEE Transactions on CAD, vol. 8, no. 7, July 1989, pp. 811-816.

[20] GAEDE R.K., M. R. MERCER and K. M. BUTLER,"CATAPULT: Concurrent Automatic Testing allowing Prallelization and Using Limited Topology," 25^{th}, ACM/IEEE Design Automation Conference, June 1988, pp. 597-600.

[21] LARABEE T., "Efficient Generation of Test Patterns Using Boolean Difference," Proceedings of the International Test Conference, August 1989.

[22] SELLERS E. F., M.Y. HSIAO and L. W. BEARSNSON, "Analyzing Errors with the Boolean Difference," IEEE Trans. Comp., vol. 17, July 1968, pp. 676-683.

[23] SMIRAT T., CHAKRADAR VISHWANI D. AGRAWAL and STEVEN G. ROTHWEILER, "A Transitive Closure Algorithm for Test Generation", IEEE Transaction on CAD, vol. 11, July 1993.

[24] AYARI BECHIR and BOZENA KAMINSKA "ROBDD Generation for General Boolean Functions", Technical Report, Ecole Polytechnique de Montréal, Mai 1993.

[25] SANJAY SRINAVASAN, GNANASEKARAN SWAMINATHAN, James H. Aylor, M. Ray Mercer, "Algebraic ATPG of Combinational Circuits Using Binary Decision Diagrams," ETC'93, pp. 240-248

[26] SHARAD C. SETH, BHARGAB B. BHATTACHARYA, VISHWANI D. AGRAWAL, "An Exact Analysis for Efficient Computation of Random-Pattern Testability in Combinational Circuits," IEEE Fault Tolerant computing, 1986, pp. 318 - 323.

[27] A.V. AHO, J. E. HOPCROFT, and J.D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison Wesley, 1974.

[28] R.E. TARJAN, "Finding Dominators in a Directed Graph," SIAM Jour. of Computing, vol. 3, 1974, pp. 62-89.

[29] D. K. SHIRACHI, "Codec Testing Using Synchronized Analog and Digital Signal," ITC, 1984, pp. 447-454.

[30] R. KRAMER "Testing Mixed-Signal Devices," IEEE Design and Test, Apr. 1987, pp. 12-20.

[31] S. MAX, "Fast, Accurate, and Complete ADC Testing," International Test Conference, 1989.

[32] K. D. WAGNER and T. W. WILLIAMS, "Design for Testability of Mixed-Signal Integrated Circuits," International Test Conference, 1988, pp.823-828.

[33] P. FASSANG , D. MULLINS and T. WONG, "Design For Testability For Mixed-Signal ASICs," IEEE Custom Integrated Circuits Conf., 1988, pp. 16.5.1-4.

[34] "P1149.4 Mixed-Signal Test Bus Framework Proposal," Panel session, IEEE International Test Conference, 1992, pp. 554 -556.

[35] A. F. ALANI, G. MUSGRAVE and A.P. AMBLER, "A Stady-State Response Test Generation For Mixed-Signal Integrated Circuits," IEEE International Test Conference, 1992, pp. 415-421.

[36] P. S. A. EVANS, M. A. AL-QUTAYRI and P. R. SHEPHERD, "A Novel Technique For Testing Mixed-Signal ICs," European Test Conf., 1991, pp. 301-306.

[37] NAIM BEN-HAMIDA and BOZENA KAMINSKA, "Analog Circuit Testing Based on Sensitivity Computation and New Circuit Modeling," IEEE International Test Conference, Oct. 1993.

[38] W. J. BUTLER and S. S. HAYKIN, "Multiparameter Sensitivity Problems in Network Theory," Proc. Institute of Electrical Engineers, Vol 117, No.12, Dec. 1970, pp 2228-2236.

[39] NAIM BEN-HAMIDA and BOZENA KAMINSKA, "Multiple Fault Analog Circuit Testing By Sensitivity Analysis," Journal of Analog Integrated Circuits and Signal Processing, Kluwer Academic Publisher, 1993, pp. 231-243.

[40] L. MILOR and V. VISAVANATHAN, "Detection of Catastrophic Faults in Analog Circuits," IEEE Trans. Computer Aided Design, Vol. CAD-8, No. 2 Feb. 1989, pp. 114-130.

[41] BECHIR AYARI and BOZENA KAMINSKA "BDD_FTEST: Fast, Backtrack-Free Test Generator Based on Binary Decision Diagram Representation," to appear in Transaction on CAD.

[42] N. NAGI and J. A. ABRAHAM, "Hierarchical Fault Modeling for Analog and Mixed-Signal Circuits," IEEE VLSI Test Symposium 1992, pp. 96-101.

[43] A. MEIXNER and W. MALY, "Fault modeling for the Testing of Mixed Integrated Circuits," International Test Conference 1991, pp. 564-572.

[44] "Standard Test Acces Port and Boundary-Scan Architecture," Sponsored by Test Technology Technical Committee of the IEEE Computer Society, Document p 1149.1/D5 (Draft), June 20, 1980.

[45] H. CHO, G. D. HACHTEL and F. SOMENZI, "Fast Sequential ATPG Based on Implicit State Enumeration," Proc. of International Test Conference, 1991.

[46] H. CHO, S.-W. JEONG, F. SOMENZI and C. PIXLY, "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation," Journal of Electronic Testing: Theory and applications, vol. 4, no. 1, Fevrier 1993.

[47] I. POMERANZ and S. M. REDDY, "On Achieving Complete Fault Coverage for Sequential Machines," IEEE transactions on Computer-Aided Design, Mar. 1994.

[48] JAEHONG PARK, CHANHEE OH and M. RAY MERCER, "Improved Sequential ATPG Using Functional Observation Information and New Justification Methods," Proc. The European Design and Test Conference, pp. 262-266, 1995.

[49] I. POMERANZ and S. M. REDDY, "The Multiple Observation Time Test Strategy," IEEE Trans. on Computer-Aided Design, Mai 1992.

[50] D. H. LEE and S. M. REDDY, "A New Test Generation Method for Sequential Circuits," Proc. of International Conference on Computer Aided Design, 1991.

[51] T. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuit," Proc. of European Design Automation Conference, 1991.

[52] M. ABRAMOVICI, M, A, BREUER and A. D. FRIEDMAN, "Digital System Testing and Testability Design", Comp. Science Press 1990.

[53] NAIM BEN-HAMIDA and BOZENA KAMINSKA, "Hierarchical Functional Testability Analysis," ETC, 1991, pp. 327-332.

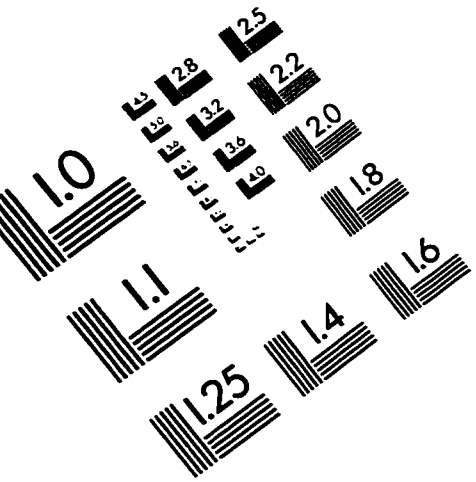
[54] H. K. LEE and D. S. HA, "HOPE: An Efficient Parallel Simulator for Synchronous Sequential Circuits," DAC, 1992, pp. 336-340.

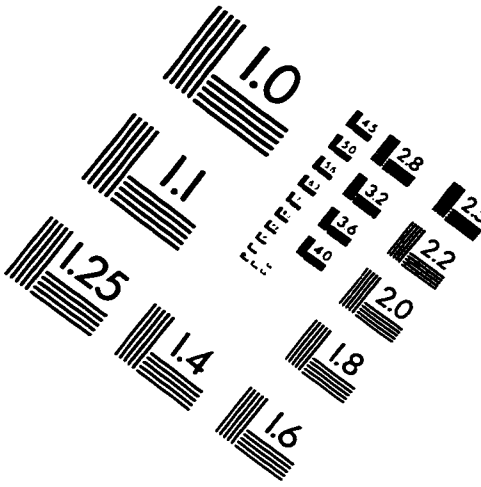
[55] G. D. HACHTEL and R.M. JACOBY, "Algorithms for Multilevel Tautology and Equivalence," IEEE International symposium on circuits and systems, Juin 1989.

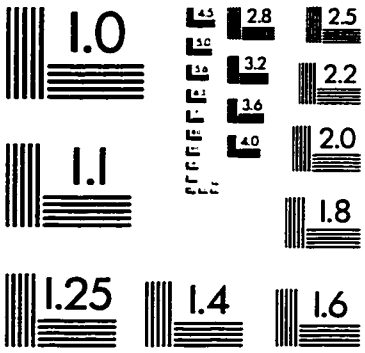
[56] R. WEI and A.L. SANGIOVANNI-VINCENTELLI, "Proteus: A Logic Verification System for Combinational Circuits", Proc. International Test Conference, 1986.

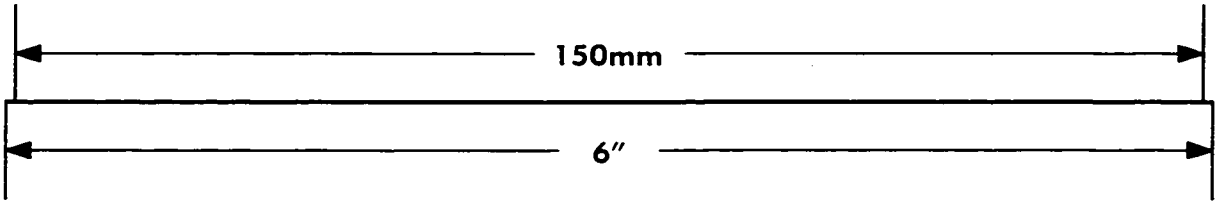
[57] B. AYARI , N. BEN HAMIDA and B. KAMINSKA, "Automatic Test Vector Generation for Mixed-Signal Circuits," European Design & Test Conference, PARIS, Marsh 1995, pp. 458-463.

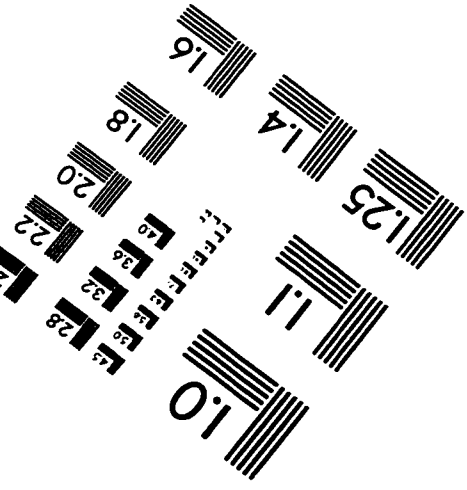
IMAGE EVALUATION
TEST TARGET (QA-3)

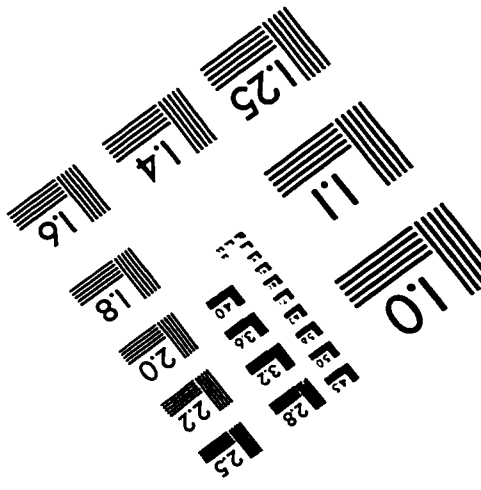












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989